REAL-TIME DATA OPERATIONS AND CAUSAL SECURITY ANALYSIS FOR EDGE-CLOUD-BASED SMART GRID INFRASTRUCTURE

BY

WENYU REN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Doctoral Committee:

Professor Klara Nahrstedt, Chair
Professor Carl A. Gunter
Professor Tarek F. Abdelzaher
Associate Professor Suleyman Uludag

ABSTRACT

The electric power grids are one of the fundamental infrastructures of modern society and are among the most complex networks ever made. Recent development in communications, sensing and measurement techniques has completely changed the traditional electric power grid and has brought us the intelligent electric power grid known as Smart Grid. As a critical cyber-physical system (CPS), Smart Grid is an integration of physical components, sensors, actuators, control centers, and communication networks.

The key to orchestrate large scale Smart Grid is to provide situational awareness of the system. And situational awareness is based on large-scale, real-time, accurate collection and analysis of the monitoring and measurement data of the system. However, it is challenging to guarantee situational awareness of Smart Grid. On the one hand, connecting a growing number of heterogeneous programmable devices together introduces new security risks and increases the attack surface of the system. On the other hand, the tremendous amount of measurements from sensors spanning a large geographical area can result in a reduction of available bandwidth and increasing network latency. Both the lack of security protection and the delayed sensor data impede the situational awareness of the system and thus limit the ability to efficiently control and protect large scale Smart Grids in time-critical scenarios.

To target the aforementioned challenge, in this thesis, I propose a series of frameworks to provide and guarantee situational awareness in Smart Grid. Taking an integrated approach of edge-cloud design, real-time data operations, and causal security analysis, the proposed frameworks enhance security protection by anomaly detection and managing as well as causal reasoning of alerts, and reduce traffic volume by online data compression. Extensive experiments by real or synthetic traffic demonstrate that the proposed frameworks achieve satisfactory performance and bear great potential practical value.

To my parents and my grandparents, for their love and support.

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my advisor Professor Klara Nahrstedt for her invaluable guidance and continuous support of my Ph.D. study. She offered me such a great opportunity to join her group and her guidance helped me in every aspect of my research and the writing of this thesis. Having learned so much from her, I could not have imagined having a better advisor.

Besides my advisor, I would like to thank my other committee members, Professor Carl A. Gunter, Professor Tarek F. Abdelzaher, and Associate Professor Suleyman Uludag for their constructive suggestions and insightful comments. Their precious inputs allowed me to greatly improve my research as well as my thesis. I am sincerely honored to have them on my doctoral committee.

I would like to thank the professors, researchers, and lab mates who gave me help and advice during my Ph.D. study. To name a few, I would like to thank Professor King-Shan Lui, Tim Yardley, Steve J. Granda, Hongyang Li, Shannon Chen, Haiming Jin, Phuong V. Nguyen, Dongjing He, Scott(He) Huang, Zhenhuan Gao, Siting Chang, Tuo Yu, Tarek Elgamal, Tianyuan Liu, Bo Chen, Zhe Yang, and Hongpeng Guo for their advice, collaboration and contribution in various research projects.

I am also grateful to all my friends who helped, encouraged, and supported me during my Ph.D. study. I feel so fortunate to have them by my side. Unfortunately, it is impossible to mention all of them so that I may only able to list some names in UIUC: Lihong Zhao, Pengkun Yang, Dan Tao, Yi Liu, Shiyan Zhang, Shengmei Xu, Siyang Xie, Lingyu Ma, Lufan Wang, Qiang Ning, and Chuchu Fan.

A very special gratitude goes out to the (DOE) Department of Energy (under Award Number DE-OE0000676 and DE-OE0000780) for their financial support and assistance.

Last but most importantly, I would like to express my deepest gratitude to my father Duoli Ren, my mother Liping Wei, my grandfather Sigen Ren, and my grandmother Ling Shao for supporting me spiritually and financially. This thesis would not have been possible without their love.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

As one of the fundamental infrastructures of modern society, electric power grids are among the most complex networks every made [1]. Over the past years, electric power grids have transformed from mostly isolated local systems to interconnected large-scale networks to deliver electricity from producers to consumers over long distances [2, 3, 4, 5]. The growing number of nodes, the increasing extent of interconnectedness, the rising variety of distributed resources, controls and loads, all contribute to the complexity of electric power grids.

Recent development in communications, sensing and measurement techniques has completely changed the traditional electric power grid. Technologies related to the Internet of Things (IoT) in the fourth industrial revolution have brought us the intelligent electric power grid known as Smart Grid. As a "digital upgrade" of the traditional electric power grid, Smart Grid employs computer, communication, monitoring, control, and self-healing technology for the following purposes [1, 6]:

- Facilitate the interconnection and operation of various distributed generating sources;
- Enhance distributed generation load balancing by grid energy storage;
- Provide the consumers with greater choice of supply and minimize their cost;
- Reduce the environmental impact;
- Increase the reliability and security of electric power delivery.

As a critical cyber-physical system (CPS), Smart Grid is an integration of physical components, sensors, actuators, control centers, and communication networks. The states of the physical components are measured and monitored by the sensors and the required operations of physical components are carried out by actuators. Via the communication networks, the control centers receive measurements from sensors and send commands to actuators, ensuring the grid operates in desired states [7].

1.1 MOTIVATION AND CHALLENGES

It is hard to orchestrate large scale Smart Grid which is a heterogeneous, widely dispersed, yet globally interconnected system. The key to monitor and control Smart Grid is to provide *situational awareness* of the system. NIST states that "the goals of situational awareness

are to understand and ultimately optimize the management of power-network components, behavior, and performance, as well as to anticipate, prevent, or respond to problems before disruptions arise" [8]. Situational awareness is based on large-scale, real-time, accurate collection and analysis of the monitoring and measurement data of the system [9].

Providing situational awareness of Smart Grid is challenging. On the one hand, connecting a growing number of heterogeneous programmable devices together introduces new security risks and increases the attack surface of the system. To make things worse, those devices are mostly resource constrained and thus do not have the capability of providing strong security protection by themselves. On the other hand, the tremendous amount of measurements from sensors spanning a large geographical area can result in a reduction of available bandwidth and increasing network latency. This in turn would hamper the timely analysis of the data and slow down the notification of time-critical events [10, 11]. Both the lack of security protection and the delayed sensor data impede the situational awareness of the system and thus limit the ability to efficiently control and protect large scale Smart Grids in time-critical scenarios.

The Supervisory Control And Data Acquisition system (SCADA) and the Wide Area Monitoring System (WAMS) are two of the most commonly used control and monitoring systems for Smart Grid. There are different challenges in providing situational awareness in these two systems and both systems deserve our attention.

- **SCADA:** The main challenge to guarantee situational awareness in SCADA is the lack of security protection. Smart Grid provides the traditional electric power grids with new functionalities and transforms the closed legacy control networks to open IP-based networks. However, new security risks are also introduced to the system at the same time. In the past, the electric power grids were considered to be secure due the limited connectivity of the system and the proprietary controls by each company. This is not true anymore since Smart Grid largely increases the connectivity of the system. Nowadays, SCADA systems have much higher exposure and larger attack surface. However, many devices and protocols in SCADA are not designed with security in mind and lack the vital security protection capabilities. The situational awareness of the system will definitely suffer if security breaches occur and are not treated quickly and properly. So, the major concern in SCADA is the security protection of it.

- **WAMS:** The main challenge to guarantee situational awareness in WAMS is the huge and increasing volume of data. In Smart Grid, WAMS has been widely accepted to provide real-time monitoring, protection, and control. WAMS is a more advanced technology than SCADA which is deployed to monitor selected critical nodes at this

moment. However, we are seeing rapid increasing deployment of WAMS and there is no sign that this trend will stop soon. Security in WAMS is less of a problem since it uses more advanced devices and protocols. However, since WAMS requires data to be captured and sent at very fast rate, the data volume in WAMS is much larger than in SCADA and delay is a much bigger problem. The huge and rapidly increasing data volume in WAMS imposes a heavy burden on the communication and storage systems and could become the bottleneck for many real-time Smart Grid applications. The delayed or dropped measurement data due to network congestions will certainly hamper the situational awareness of Smart Grid.

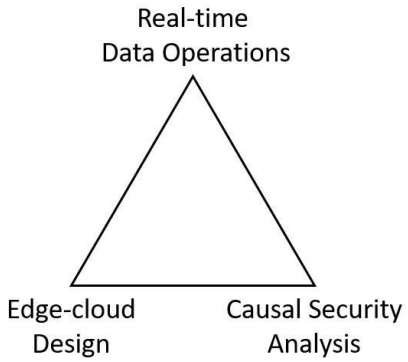


Figure 1.1: Triangle of situational awareness in Smart Grid

In this thesis, we argue that there are three necessary requirements to provide situational awareness to Smart Grid. As shown in Figure 1.1, the three requirements include *real-time data operations*, *causal security analysis*, and *edge-cloud design*. The three requirements form a triangle and the situational awareness of Smart Grid can only be achieved when all three of them are considered.

Measurement and monitoring data of Smart Grid is generated, gathered, and sent to the control center of Smart Grid for analysis all the time. For the data to be useful, one important requirement is that the data operations in the system must be conducted in real time. Since the term situational awareness is defined as "up-to-the-minute cognizance or awareness required to move about, operate equipment, or maintain a system" [12], the timely acquisition and analysis of sensor data is critical to guarantee situational awareness of the grid. Also, real-time event processing is essential for Smart Grid, and cyber-physical applications, running in the control centers, are usually subject to strict latency constraints. Therefore, data operations in Smart Grid are time sensitive and we should always keep time overhead and latency in mind for the rest of this thesis.

Besides guaranteeing the timeliness of data, the security of Smart Grid is also vital for the situational awareness of Smart Grid. And as we mentioned before, the goal of situational awareness is more than just to know the state of the system but also to understand why the system is in that state. In terms of security, knowing what anomalies are happening in the system is not enough. Understanding why they happen is also of critical importance. If only intrusion detection is deployed and the causes and consequences of the events are not identified, it is hard or impossible for the operator to quickly digest the events and react to them. Even if the events related to attacks are detected quickly, the entire system could still suffer when the true reasons for the events remain unexplained and the operator fails to resolve the attacks promptly. Therefore, to guarantee situational awareness of Smart Grid, it is necessary to provide causal security analysis of the system which includes not only the detection but also the reasoning of anomalies.

To provide situational awareness, another novel paradigm called edge computing (also known as fog computing) [10, 13, 14] also needs be exploited. Edge computing is an architecture that uses networking edge devices to carry out computing services closer to end devices. By bringing more computing power to the outer edges of the network, which are closer to the data sources, edge computing is able to process and analyze the massive data from different kinds of end devices faster. This, in turn, can help to reduce the amount of data to be transmitted to the data centers and to provide services with faster response as well as greater quality [15]. For Smart Grid, the control centers host the cloud and the substations include the edges. By using an edge-cloud design in Smart Grid, inspection and preliminary analysis of SCADA traffic can be placed close to end devices to provide timely information to power grid operators. The huge data volume in WAMS can also be reduced if data compression operation is placed at the edge. Therefore, an edge-cloud design is essential in providing situational awareness to Smart Grid by helping to achieve a faster actuation and response of the Smart Grid system to events, a better utilization of the communication bandwidth, and an increase of reliability and scalability [16].

1.2 THESIS STATEMENT

I claim that the following thesis statement is true.

To provide Smart Grid with situational awareness, we need an integrated approach of edge-cloud design, real-time data operations, and causal security analysis.

1.3 THESIS OVERVIEW

In this thesis, I design four frameworks, deployed in Smart Grid, to achieve the objective of situational awareness. In this section, I give an overview of each of them and explain how they help to achieve the objective.

1.3.1 OLAF: Operation-Level Traffic Analyzer Framework for SCADA System

As we mentioned in Section 1.1, the current SCADA systems in Smart Grid are facing increasing security risks. They are primarily protected at the perimeter level with firewalls at the boundary of the networks but are not equally protected against internal attacks. To provide end-to-end security against both external and internal attacks, both the end host devices and the network need to be secured. However, the end host devices in SCADA such as Programmable Logic Controllers (PLCs) and Remote Terminal Units (RTUs) are resource-constrained devices which do not have the ability to provide security analysis and protection by themselves [17]. Therefore, in the current architecture, the control centers are responsible for both the device status analysis and network traffic analysis. Data from the measurement devices is transmitted to the control center for processing and analysis. As the number of deployed measurement devices grows, it is increasingly harder for the current architecture to provide up-to-date situational awareness to the power grid operators.

To address the above problem in Smart Grid SCADA networks, my approach in this work [18] is the design of an edge-based, extensible, and efficient operation-level traffic analyzer, called OLAF. OLAF is able to collect, aggregate and analyze the statistics in network packets from both the flow level for network traffic analysis and the operation level for device status analysis. I deploy OLAF close to the end hosts in the edge of SCADA network to provide the operators with more up-to-date situational awareness of the whole system and warn them of potential breaches more promptly. This work has been published in **IEEE SmartGridComm 2016** [18].

1.3.2 ISAAC: Intelligent Synchrophasor Data Real-Time Compression Framework for WAMS

As stated in 1.1, WAMS have been widely accepted to provide real-time monitoring, protection, and control of power systems. WAMS's capability to support real-time decision-making applications is based on the emerging and development of Phasor Measurement Units (PMUs). Since PMUs have very high sampling rates and usually multiple data channels, the volume of measurements collected is huge. And we can surely expect a multi-fold expansion

in the already large volumes of data in WAMS due to the higher sampling rate of modern PMUs, the increase in the number of PMUs and measurements per PMU. The huge and rapidly increasing data volume in WAMS imposes a heavy burden on the communication and storage systems and could become the bottleneck for many real-time smart grid applications. If not handled carefully, the huge data volume in the communication system could result in frequent and severe congestion. The situational awareness of the system could suffer a lot from the extremely long delays or high packet loss rates that follow the congestion.

In this work, I propose an Intelligent Synchrophasor dAta reAl-time Compression framework, named ISAAC, to be deployed at the edge of WAMS. Combining the Principal Component Analysis (PCA) and Discrete Cosine Transform (DCT), ISAAC has the capability of largely improving the efficiency of communication and storage systems via data compression while maintaining strong data fidelity. This work has been published in **IEEE SmartGridComm 2017** [19].

1.3.3 EDMAND: Edge-Based Multi-Level Anomaly Detection for SCADA Networks

As discussed in Section 1.3.1, I proposed a light-weighted operation-level traffic analyzer, named OLAF, to provide preliminary analysis of SCADA. However, that is not enough to guarantee situational awareness and a more thorough monitoring and analysis are required. Based on different analysis granularity, data in SCADA network traffic generally can be divided into three levels: transport level, operation level, and content level. Transport level data refers to statistics in IP headers and transport protocol headers. Operation level data refers to operation statistics in ICS protocols. Content level data refers to measurement statistics from field devices. Monitoring and event detection of only one or two of the three levels is not enough to detect and reason about attacks in all three levels. Also, data in each level has its own characteristics, which requires distinct methods to deal with.

In this work, I develop an edge-based multi-level anomaly detection framework for SCADA networks, named EDMAND. EDMAND is located inside the remote substations, which are the edges of the SCADA network. It contains a multi-level anomaly detector to monitor all three levels of network traffic data passing by. Appropriate anomaly detection methods are applied based on the distinct characteristics of data in various levels. The concept of confidence is introduced into the anomaly detection process and confidence scores are assigned to generated alerts. Also, the generated alerts are aggregated and prioritized to benefit further analysis before being sent back to control centers. This work has been published in **IEEE SmartGridComm 2018** [20].

1.3.4 CAPTAR: Causal-Polytree-based Anomaly Reasoning for SCADA Networks

To promote the security of SCADA systems, intrusion detection systems (IDSs) are increasingly deployed by SCADA operators. The main objective of IDSs is to monitor the system, detect suspicious activities caused by intrusion attempts, and report alerts to the system operators. However, as we mentioned in Section 1.1, only knowing what anomalies are happening in system without understanding why they happen is definitely not enough to guarantee situation awareness. Traditional IDSs continuously generate tremendous number of alerts without further comprehending them. Drowned in an ocean of unstructured alerts mixed with false positives, SCADA operators are almost blind to see any useful information. Due to the high volume and low quality of the alerts, it becomes a nearly impossible task for the operators to figure out the complete pictures of the attacks and take appropriate actions in a timely manner.

Therefore, there is a need for an efficient system to aggregate redundant alerts from IDSs, correlate them in an intelligent manner, and discover attack strategies based on domain knowledge. My previous work EDMAND, located at the edges of the SCADA network, detects anomalies in multiple levels of the network, and sends aggregated and prioritized meta-alerts to the control center. In this work, I present a causal-polytree-based anomaly reasoning framework for SCADA networks, named CAPTAR. CAPTAR resides in the control center of the SCADA network and takes the meta-alerts from EDMAND as input. CAPTAR correlates the alerts using a naive Bayes classifier and matches them to predefined causal polytrees. Utilizing Bayesian inference on the causal polytrees, CAPTAR is able to reveal the attack scenarios from the alerts and produces a high-level view of the security state of the protected SCADA network. I am planning to submit this work to **IEEE SmartGridComm 2019**.

1.4 THESIS CONTRIBUTION

The contributions of this thesis are as follows:

- Before I proposed OLAF, all the security analysis for SCADA networks happened in the control center. Although edge-computing was already a popular paradigm at that time, not many works had applied it to the Smart Grid domain. OLAF is one of the earliest works to introduce the concept of edge-computing as well as smart edge devices into Smart Grid. It is also one of the first works to place anomaly detection at the edge of SCADA networks. By pushing security analysis to the edge, faster detections can be achieved for anomalies and reactions can also be performed more promptly.

- At the time I was developing ISAAC, issues caused by the huge data volume in WAMS had not drawn much attention from researchers yet. Of all works that focus on alleviating the burden from the data volume, most of them only worry about storage problems and use offline compression to solve them. ISAAC is one of the first works that identify the problem that huge data volume could cause on communication networks and use online compression to target the problem. The challenges and requirements of online compression in WAMS are different from those of offline compression and ISAAC is the first work to discuss and target them.

- In EDMAND, we divide network traffic data in SCADA into three levels: transport, operation, and content levels. Previous works mostly focus on only one or two of the three levels of data. In EDMAND, we cover all three levels and apply appropriate anomaly detection mechanisms to data in each level based on their distinct characteristics, which is necessary to see the whole picture of the attack. Also, EDMAND is one of the first works to introduce the concept of confidence into the anomaly detection process and assign confidence scores to generated alerts.

- Before CAPTAR, most works in the area of SCADA security stop at anomaly detection. The true reasons that caused the anomalies are left unexplained. In CAPTAR, we go one step further than that and use alert correlation and causal reasoning to understand the causes of the anomalies. Attack scenarios can be revealed from the alerts and a high-level view of the security state of the SCADA network can be produced. This prevents the operator from being overwhelmed in the huge number of low-quality alerts and provides situational awareness that is explainable.

1.5 THESIS ORGANIZATION

Chapter 2 introduces some background knowledge about Smart Grid to be used in the rest of this thesis. In each of Chapter 3-6, I will describe in more details one of the four aforementioned works, describing its design and elaborating on the evaluation performance. To be more specific,

- In Chapter 3, to provide end-to-end security for SCADA systems in Smart Grid against both external and internal attacks and enhance up-to-date situational awareness, I propose an edge-based, extensible, and efficient operation-level traffic analyzer, called OLAF.

- In Chapter 4, to alleviate the burden of huge and rapidly increasing data volume in WAMS and prevent the impairment of situational awareness by frequent and severe congestion, I propose an intelligent synchrophasor data real-time compression framework, named ISAAC.

- In Chapter 5, to provide Smart Grid SCADA systems with more comprehensive situational awareness, I develop an edge-based multi-level anomaly detection framework, named EDMAND, to monitor and detect anomaly of all transport, operation, and content levels of SCADA network traffic.

- In Chapter 6, to offer explainable situational awareness to Smart Grid, I present a causal-polytree-based anomaly reasoning framework for SCADA, named CAPTAR, to correlate alerts from EDMAND in an intelligent manner and discover attack strategies based on domain knowledge.

In Chapter 7, I conclude this thesis and have a discussion of the generalization about this thesis. Finally, I share some lessons learned and provide some directions of future research in Chapter 8.

CHAPTER 2: BACKGROUND

In this chapter, I will introduce some background knowledge about Smart Grid which will be used in the following chapters of this thesis. In Section 2.1, I describe the basic structure of the electric power system. Two commonly used systems in Smart Grid, namely SCACA and WAMS, are briefly introduced in Section 2.2. Then two popular industrial control protocols, Modbus and DNP3, are mentioned in Section 2.3.

2.1 ELECTRIC POWER SYSTEM

As a cyber-physical system, Smart Grid consists of both physical and cyber parts. The physical part of Smart Grid is the electric power system. As it is shown in Figure 2.1, the basic structure of the electric power grid has four main components: generation, transmission, distribution, and customer.

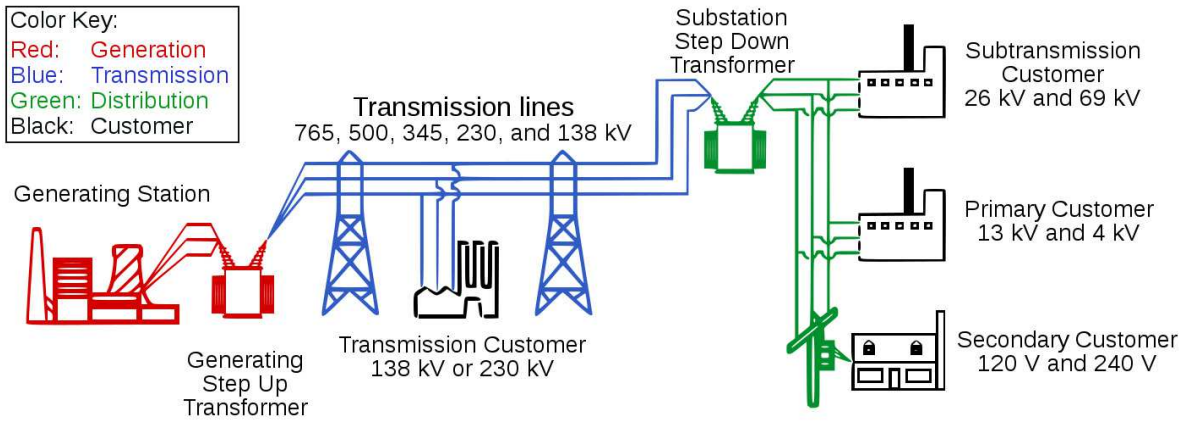


Figure 2.1: Basic structure of the electric power system [21]

In the generating stations, also referred to as power plants or power stations, electricity is produced at lower voltages (10kV to 25kV). Most generating stations burn fossil fuels such as coal, oil, and natural gas and use generators to turn mechanical power into electrical power. Others use nuclear power, and there is an increasing trend of using renewable sources such as solar, wind, hydro power, geothermal, etc. Near the generating stations, step-up transformers increase the voltage of electricity from the generating stations for transmission.

In the transmission component, electricity is transmitted in bulk from generating stations to substations over transmission lines. Transmission lines are usually operated at high voltages (i.e., 115kV or above) to reduce the energy loss over long distances. Compared with

low-voltage transmission, high-voltage transmission allows for less energy loss from conductor heating and delivers a larger proportion of the generated power to the substations, and thus achieves economic power transmission. Transmission lines are interconnected at switching stations and substations to form a network called a power grid.

The distribution component is the final stage of electric power delivery which distributes electricity from the transmission system to individual customers. Distribution substations use step-down transformers to lower the transmission voltage to intermediate voltage levels (2kV to 35kV). The primary distribution system feeds larger industrial and commercial customers. It also carries the intermediate voltage power to small substations closer to residential end customers. At these substations, voltage is again lowered by transformers to a service voltage (i.e., 120V or 240V) and then electricity is carried by the secondary distribution system to customers for residential use.

2.2 SCADA AND WAMS

What makes Smart Grid different from traditional power grids is its cyber part. The cyber part has several systems that are widely deployed, including the Supervisory Control And Data Acquisition system (SCADA), the Wide Area Monitoring System (WAMS), and the Advanced Metering Infrastructure (AMI). Among these three systems, SCADA and WAMS are responsible for the generation, transmission and distribution components whereas the AMI is responsible for the customer component. In this thesis, I focus on SCADA and WAMS since they are more critical systems and are highly related to the situational awareness of Smart Grid.

A SCADA system is a common industrial control system which is used to collect data from sensors located at remote sites and to issue commands from a central site for control purpose. The simplified architecture of a typical SCADA system is shown in Figure 2.2. The major components in SCADA system include the Master Terminal Units (MTUs) in the control centers, field controllers in the remote substations and the communication network that connects them. The MTU, also referred to as SCADA server or supervisory controller, is the core of the SCADA system. It is a server responsible for communicating between the field controllers and the human-machine interface (HMI) software running on operator workstations. The field controllers mainly consist of Remote Terminal Units (RTUs) and Programmable Logic Controllers (PLCs). Further connected to sensors and actuators, these controllers are responsible for transmitting telemetry data to the MTU and controlling connected actuators by messages from the MTU [22]. The communication network connects the MTU to RTUs and PLCs by radio, wire, or optical fiber connections. Industrial control

protocols are used in the communication network and two of the most commonly used ones will be introduced in Section 2.3.

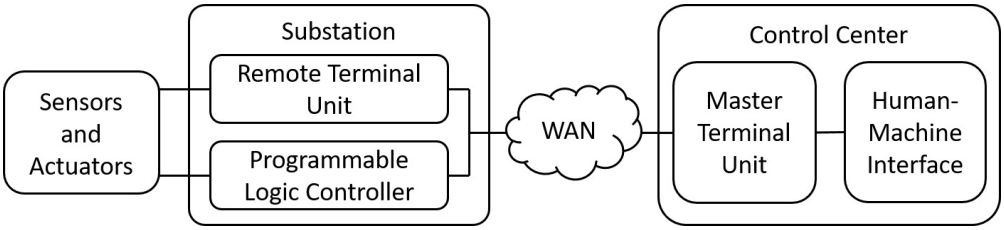


Figure 2.2: Simplified architecture of SCADA

WAMS is a new advanced measurement technology to collect information. WAMSs ability to support dynamic monitoring of the system conditions over large areas is allowed by the emerging and development of a new data acquisition technology of phasor measurement. A simplified architecture of WAMS is shown in Figure 2.3. Phasor Measurement Units (PMUs) are installed in selected substations to measure the connected bus bars or power lines. PMUs can measure frequency, current, and voltage at a rate of 30 Hz or higher. The generated measurements are called synchrophasors, namely synchronized phasors, since they contain both magnitudes and phase angles, and are precisely time-synchronized by the GPS technology. The synchrophasor measurements from the PMUs are then transmitted to Phasor Data Concentrators (PDCs), where they are correlated and fed out as a single stream. The time-aligned measurements are forwarded via WAN to the control center and usually further concentrated by the PDC there. Finally, the measurements are consumed by the WAMS applications in the control center. The phasors measured by PMUs at the same instant can be seen as a snapshot of the system condition. By comparing the snapshots with each other, monitoring of the dynamic as well as steady state of the system is achieved.

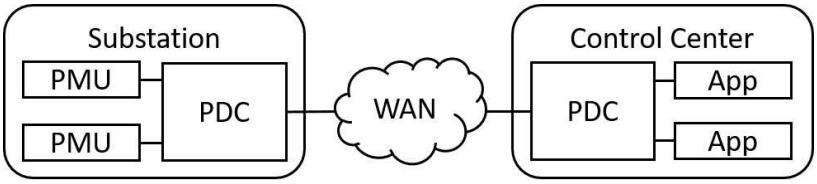


Figure 2.3: Simplified architecture of WAMS

WAMS is a more advanced technology than SCADA. SCADA can only provide non synchronous information of the steady state of system with low sampling rate while WAMS allows us to monitor the dynamic state of system synchronously in more elaborate time scale. Since WAMS requires data to be captured and sent at very fast rate, the data volume in WAMS is much larger than in SCADA and delay is a much bigger problem in WAMS.

Nowadays, both SCADA and WAMS are utilized in Smart Grid. SCADA is used as the major technology and WAMS is applied to selected critical nodes. Therefore, both SCADA and WAMS are considered and discussed in this thesis.

2.3 MODBUS AND DNP3

In this section, I will briefly introduce two industrial control protocols commonly used in SCADA: Modbus and DNP3. All our works for SCADA in the following chapters support these two protocols. Note that common protocols used in WAMS include IEEE C37.118 and IEC 61850. Since they do not affect the online data compression in WAMS which I focus on, I will not describe protocols in WAMS in more details.

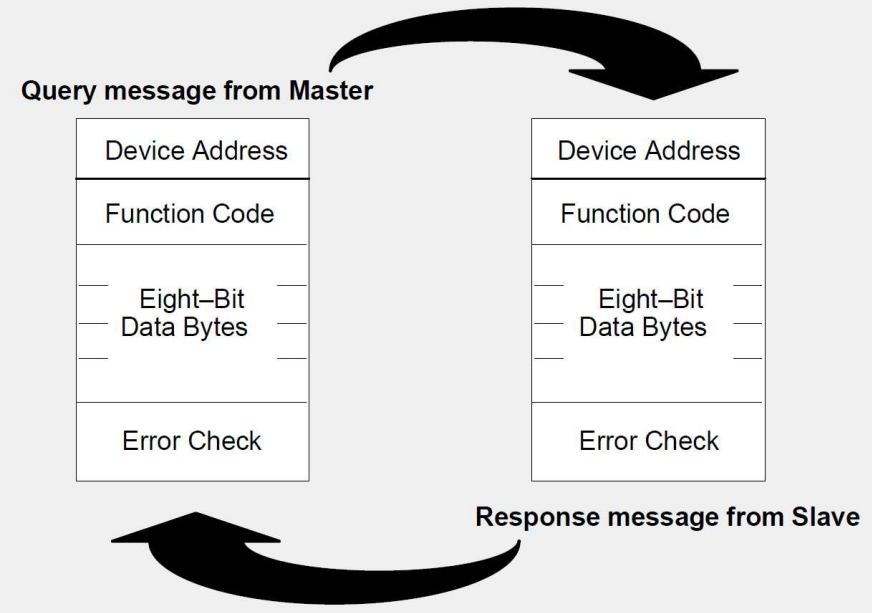


Figure 2.4: Modbus query–response cycle [23]

Modbus [24] is an application layer messaging protocol, which provides master-slave communication between devices connected on different types of buses or networks. Usually, only one device (the master) initiates queries while other devices (the slaves) respond by sending the requested data or by taking the requested action [23]. In SCADA, MTU is a typical master device and typical slaves include RTUs and PLCs. As it is shown in Figure 2.4, for each Modbus transaction, the master originates a query message and expects a response from a slave device. Similarly, a response message is constructed by the slave and returned to the master when the slave receives a query. The master's query consists of the device address, a function code, any data to be sent, and an error check field. The function code

in the query tells the slave what kind of action to perform. Any additional information required by the slave to perform the function is included in the data bytes. And the slave is able to verify the integrity of the query by using the error check field in the message. The response message from the slave shares the same structure of the query message. In a normal response, the function code is just an echo of the query function code and the data bytes contain the requested value such as register values or status. If there is an error, the function code is set to an exception function code indicating an error response and the data bytes contain an error code that further describes the error.

Modbus's data model relies on a series of tables that have distinguishing characteristics. The four primary tables are listed in Table 2.1. Modbus supports individual selection of 65536 data items for each of the primary tables. The function codes of Modbus can be categorized into three types: public, user-defined, and reserved function codes. The public function codes are well defined function codes which are guaranteed to be unique. Some of the most commonly used public function codes are the read and write operations of primary tables and are shown in Table 2.2. The user-defined function codes are not supported by the specification and are implemented by users. There is no guarantee that the use of user-defined function codes will be unique. The reserved function codes are used by some companies for legacy products and are not available for public use.

Primary Tables	Object Type	Type of	Comments
Discrete Input	Single bit	Read-Only	This type of data can be provided by an I/O system.
Coils	Single bit	Read-Write	This type of data can be altered by an application program.
Input Registers	16-bit word	Read-Only	This type of data can be provided by an I/O system.
Holding Registers	16-bit word	Read-Write	This type of data can be altered by an application program.

Table 2.1: Data model of Modbus [23]

DNP3 [25], which represents the Distributed Network Protocol Version 3, is a standards-based communications protocol used between components in industrial control systems such as SCADA. To be more specific, DNP3 is responsible for the exchanging of data and control commands between master and outstation. The term outstation denotes remote computers in the field and the term master represents computers in the control centers. In SCADA, MTU is a typical master computer and typical outstation computers include RTUs and PLCs. Using DNP3, the master station issues control commands to outstation computers and outstation computers gather data for transmission to the master. As it is shown in

Code	Name
01	Read Coils
02	Read Discrete Inputs
03	Read Holding Registers
04	Read Input Registers
05	Write Single Coil
06	Write Single Register
15	Write Multiple Coils
16	Write Multiple Registers
23	Read/Write Multiple Registers

Table 2.2: Commonly used public function codes in Modebus [24]

Figure 2.5, DNP3 is a layered protocol that is based on the Open Systems Interconnection model (OSI model). DNP3 supports application layer, pseudo-transport layer, and data link layer. The application layer bridges the DNP3 user's code with the lower layers and provides standardized functions and data formats for the efficient transmission of control commands and data values. The pseudo-transport layer is responsible for disassembling large application layer fragments into data-link-layer-sized units for transmission and reassembling them back into the original application fragment on reception. The data link layer, lying between the pseudo-transport layer and the physical media, provides station addressing, data fragmentation, frame synchronization, link control, and error detection.

In Figure 2.5, the series of square blocks at the top of the outstation denotes the data stored in its database and output devices. Each data type is structured as a separate array. The binary input array contains states of physical or logical Boolean devices. The analog input array contains analog input quantities that the outstation measured or computed. The counter array contains ever increasing count values. The control output array contains physical or logical on-off, raise-lower and trip-close points. The analog output array contains physical or logical analog quantities such as those used for setpoints. As it is shown in Figure 2.5, the DNP3 master shares similar input data types including binary, analog and counter. These values are used by the master for system state display, closed-loop control, alarm notification, billing, etc.

Some of the most commonly used functions in DNP3 are listed in Table 2.3. For example, let us consider read operations. The master's user layer formulates its request for data from the outstation by telling the application layer function code 1 to perform, and by specifying the data types it wants from the outstation. The application layer then passes the request down through the transport layer to the link layer that, in turn, sends the message to the outstation. The link layer at the outstation checks the frames for errors

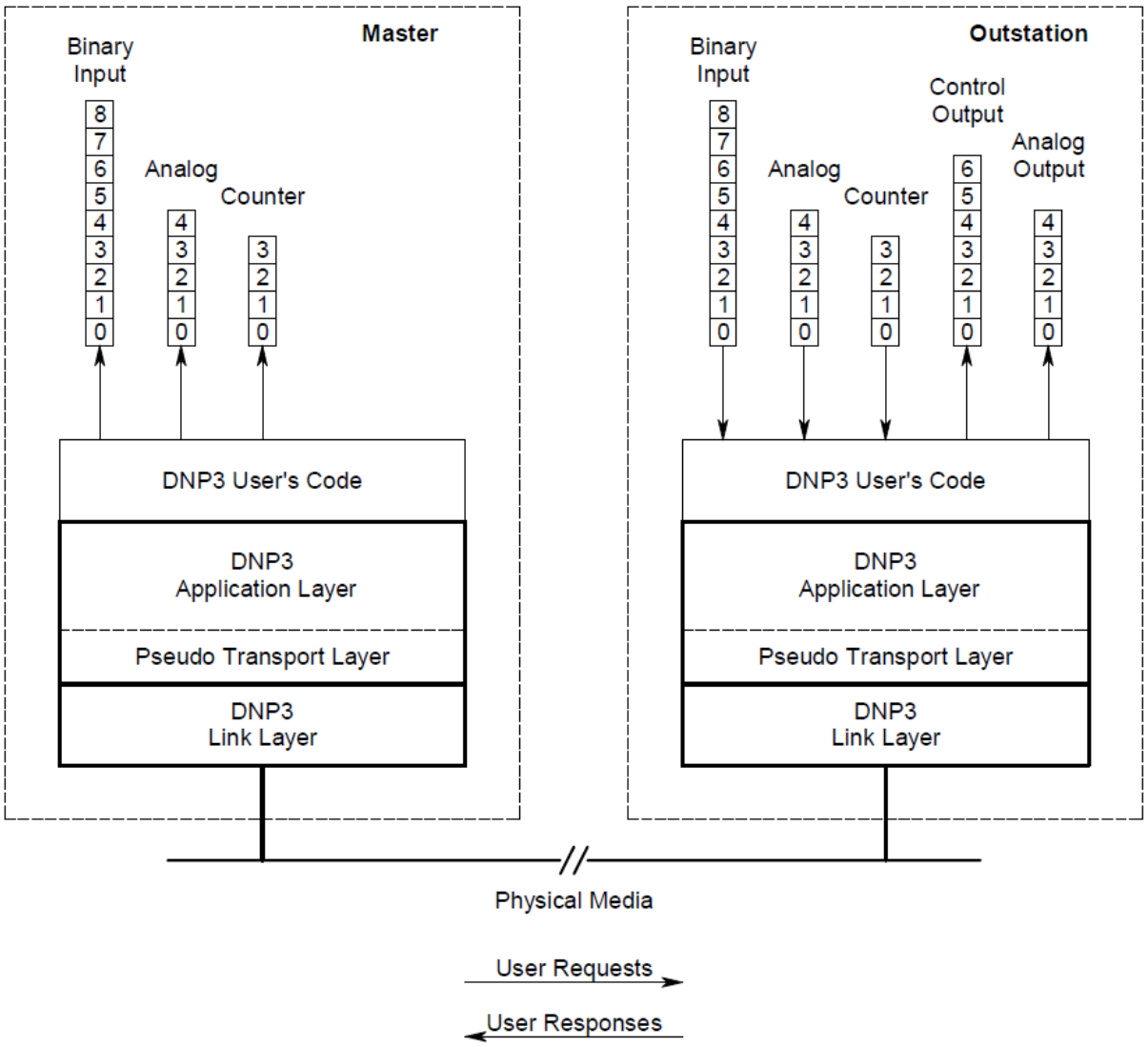


Figure 2.5: Master–outstation relationship and layering of DNP3 [26]

and passes them up to the transport layer where the complete message is assembled in the outstation's application layer. The application layer then tells its user layer what data were requested. Responses work similarly, in that, the outstation's user layer fetches the desired data and presents it to the application layer. Data is then passed downward, across the communication channel and upward to the master's user layer. Some outstations are also able to spontaneously transmits a response without having received a specific request for the data. These unsolicited responses are utilized to transmits changes at outstations as soon as possible rather than waiting for another master station polling cycle. Also, whenever the master receives an application layer fragment from an outstation, it sends a confirmation to the corresponding outstation.

In this thesis, I am considering Modbus on TCP. Therefore, both Modbus and DNP3 in this thesis have a transport layer and an internet layer and rely on the TCP/IP protocol suite.

Message Type	Code	Name	Brief Description
Confirmation	0	CONFIRM	Master sends this to an outstation to confirm the receipt of an Application Layer fragment.
Request	1	READ	Outstation shall return the data specified in the request.
Request	2	WRITE	Outstation shall store the data specified in the request.
Response	129	RESPONSE	Master shall interpret this fragment as a response to a request sent by the master.
Response	130	UNSOLICITED_ RESPONSE	Master shall interpret this fragment as an unsolicited response that was not prompted by an explicit request.

Table 2.3: Commonly used function codes in DNP3 [25]

CHAPTER 3: OLAF: OPERATION-LEVEL TRAFFIC ANALYZER FRAMEWORK FOR SCADA SYSTEM

3.1 INTRODUCTION

As we mentioned in Section 1.1, for the Smart Grid SCADA systems nowadays, various security mechanisms (e.g., firewalls and gateways) are applied to the boundary of the infrastructure to inspect and secure the information exchanged with external entities. However, data within SCADA networks, gathered by the internal field measurement devices, is usually not visible to the operators and not secured at the same security levels as communication with external entities. Smart Grid wide-area networks connect Smart Grid substation field networks with operational control centers, and these networks open possibilities for potential attacks, unless they embed end-to-end security mechanisms and policies. Even if we secure communication paths, insider attacks are possible due to spear phishing, USB, network-based malware proliferation [17], etc. As a result, it is a must for the utilities to go beyond guarding the external boundary of Smart Grid SCADA networks and begin inspecting and protecting internal Smart Grid SCADA networks at all levels.

To provide end-to-end security in the network system, both the end host devices and the network need to be secured. In general purpose network such as the Internet, the end hosts usually have their own security analysis and protection mechanism. Therefore, the analyzers designed for general purpose network [27, 28, 29, 30] only need to provide network analysis capability, i.e., flow/packet or application level traffic analysis. In Smart Grid SCADA network, the end hosts are control centers and field controllers such as Programmable Logic Controllers (PLCs) and Remote Terminal Units (RTUs). Since the field controllers are resource-constrained devices which do not have enough memory and computing resources to support the addition of security capabilities [17], the control centers are usually responsible for both the device status analysis and network traffic analysis. In the current architecture, data, collected by the measurement devices, is transmitted to the control center for processing and analysis. As the number of deployed measurement devices grows, it is increasingly harder for the current architecture to satisfy the real-time needs of protection and control applications. Therefore, the current Smart Grid SCADA networks cannot provide up-to-date situational awareness to the power grid operators. However, supervising device functionality, detecting networking anomalies and preventing potential problems in substations, all rely on situational awareness. The lack of up-to-date situational awareness may result in huge losses during security breaches.

Our approach to the problem in Smart Grid SCADA networks is the design of an edge-

based, extensible, and efficient operation-level traffic analyzer, called OLAF, that has the capabilities of both flow-level network traffic analysis and operation-level device status analysis. OLAF is able to collect, aggregate and analyze the statistics in network packets from both the flow level for network traffic analysis and the operation level for device status analysis. For network traffic analysis in the flow level, OLAF is able to track which two hosts are communicating. For device status analysis in the operation level, the analyzer is able to track status information of the utilized industrial control systems protocols (e.g., Modbus and DNP3), ongoing operations (e.g., read or write), and the targets of those operations(e.g., indexes of coils or registers that carry values of the operation). OLAF collects, aggregates and stores these meta data statistics in efficient data structure. Then, it inspects those aggregated data to perform anomaly detection.

We deploy OLAF close to the end hosts in the edge of SCADA network to provide up-to-date situational awareness of network traffic and operational device status to the power operator. OLAF provides the situational awareness by promptly extracting, aggregating, displaying and analyzing the control information in network packet headers as well as the encapsulated data. By providing both network and device analysis ability close to field controllers, we are able to give the power operator more up-to-date view of the whole system and warn them of potential breaches more promptly.

This chapter is structured as follows: We review the related work in Section 3.2. In Section 3.3, we introduce the network architecture of the Smart Grid control systems and describe the design challenges and our approaches. In Section 3.4, we present an overview of the analyzer design. Both time overhead and performance evaluation of OLAF is shown in Section 3.5 and we conclude the chapter in Section 3.6.

3.2 RELATED WORK

Traffic analyzer is the main approach to provide network traffic analysis for general purpose networks. There are many works [27, 28, 29, 30] aimed at designing network profiler and traffic analyzer. However, to our knowledge, none of the existing analyzers for general purpose networks provide operation-level device analysis. And operation-level device analysis is crucial to Smart Grid SCADA networks, since compromised devices can send malicious operations or fake measurement data which could cause huge damage to the entire system.

There are also works that build analyzers for SCADA systems. Different approaches include traffic filtering systems [31, 32], Bloom-filter-based/model-base/machine-learning-based intrusion detection [33, 34, 35, 36], SCADA Intelligence Gateway [37] and a fine grained analysis of packet content [38]. Our work differs from theirs in that instead of just

extracting and analyzing all the features separately, we factor the features into multiple levels and store our statistics in a tree structure. We have also designed a threshold-based anomaly detection algorithm utilizing the tree structure. The tree structure not only allows us to efficiently store and access the statistics, but also gives us the ability to easily change the granularity of our analysis and inspection.

3.3 PRELIMINARIES

In this section, we first introduce the Smart Grid network architecture. Then we describe the design problems of the analyzer together with our approaches.

3.3.1 Network Architecture

The simplified network architecture of Smart Grid SCADA system [17] is shown in Figure 3.1. The communications are between the MTU in the control center and field controllers within the substations. The field controllers are PLCs and RTUs which are further connected to sensor and actuators. They forward data measurements to control centers and receive control commands from control centers. The control center, on the other hand, is responsible for processing the forwarded data and issuing control commands.

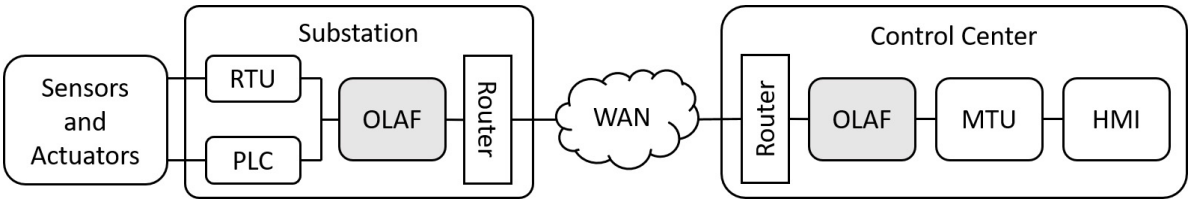


Figure 3.1: Smart Grid SCADA network architecture and OLAF

To perform analysis of the Smart Grid SCADA network traffic and device status, our analyzers are placed at the boundary of the WAN at both control center and substation ends, as it is shown in Figure 3.1. To be more specific, the uplink of the substation switch is connected to the input interface of the analyzer and the output interface is connected to the input of the router, making the analyzer inline to all communications. A similar type of connection is done also on the other side at the control center. Once the analyzers are physically connected inline, all traffic that was previously being communicated now transits through the analyzer and is subject to inspection and analysis.

3.3.2 Design Challenge

While designing the analyzer, there are many challenges we need to deal with. Two of them are crucial to our analyzer and are discussed as follows:

- ***Extensibility***[1]: As the Smart Grid SCADA networks evolve, smarter devices and new industrial control protocols are deployed. Designing the analyzer to be easily extensible to handle new operations of new devices using new protocols is of great importance. The software design concept, modularity, is our approach to this problem. OLAF is constructed by different modules with different functionalities (e.g., collection, aggregation, analysis). To support the analysis of new operations from new protocols, we can simply update corresponding modules or add new modules instead of redesigning the entire analyzer.

- ***Efficiency***: Since OLAF is going to deal with real-time traffic in the Smart Grid SCADA network and provide prompt analysis and inspection of the packet, time efficiency is crucial to our analyzer. The collection of statistics needs to be done at the communication line speed and the analysis also needs to be done fast enough to be meaningful to power grid operators. To be efficient, the analyzer should collect sufficient amount of statistics at the minimum cost. We achieve this by factoring the statistics into multiple levels and adjusting the number of used levels according to requirements of metadata accuracy as well as time and storage overhead.

3.4 DESIGN OVERVIEW

In this section, we present an overview of the analyzer system design. The modular structure of OLAF is shown in Figure 3.2. OLAF consists of 4 modules: (1)*Statistics Collector*, (2)*Statistics Aggregator*, (3)*Anomaly Detector*, (4)*Pattern-based Identity Recognizer*.

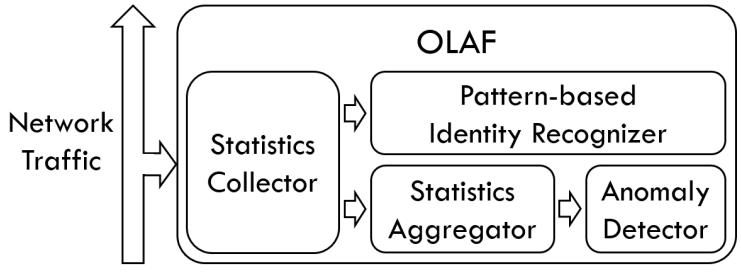


Figure 3.2: Modular Structure of OLAF

[1]Here we only discuss the extensibility to new functionalities.

The Statistics Collector examines the network packets and collects the flow-level and operation-level statistics needed by the network traffic and device status analysis. It then provides the inputs for the Statistics Aggregator and the Pattern-based Identity Recognizer. The Statistics Aggregator aggregates the statistics and sends them to the Anomaly Detector for further analysis and anomaly detection. The Pattern-based Identity Recognizer identifies the type of the traffic source and destination by monitoring certain request and response patterns in the statistics provided by the Statistics Collector. The four modules will be described in more detail in the following subsections.

3.4.1 Statistics Collector

For each network packet that goes into the collector, there are 6 levels of statistics we want to collect, which are shown in Table 3.1. Levels 1-2 are flow-level statistics that are used for network traffic analysis, while levels 3-6 are operation-level statistics that are used for device status analysis. Note that not all 6 levels of statistics necessarily exist for each packet. For example, a transport control packet (e.g., TCP ACK) does not have an industrial control protocol (e.g., Modbus) header and only has the upper two levels (e.g., levels 1 and 2) of the statistics. On the other hand, a Modbus request packet to read a specific register has all 6 levels of statistics. Another thing worth noticing is that the Unit ID (UID) is a protocol specific additional address used to differentiate aggregated data or devices that do not have IP addresses. For example, the substation could aggregate the measurement data from IEDs and the control center will communicate with the substation instead of each IED to collect those data. In this case, all the packets will have IP addresses of the control center and the substation. To differentiate data from different IEDs, the power operator will assign different additional addresses to each IED and use those as identifiers.

Level	*Subject*
1	Sender of the packet
2	Receiver of the packet
3	Protocol that the packet uses for industrial control (e.g., Modbus or DNP3)
4	Unit ID (UID) that is used by the protocol to identify different devices
5	Function (e.g., read or write) that the packet conducts in its protocol
6	Target of the function (e.g., which coil or register)

Table 3.1: 5 levels of statistics

In the Statistic Collector, each packet header will go through three metadata extractors in order. The three extractors will extract the statistics of levels {1, 2}, {3, 4, 5} and

6, respectively. The first extractor is a general one which extracts *sender* and *receiver* information. The other two, on the other hand, are protocol and device specific and are responsible for extracting {*protocol*, *UID*, *function*} and *target*, respectively. Currently, we have extractors for two industrial control protocols, DNP3 and Modbus. An *item_gen* event will be triggered after the packet is processed by the last extractor, which contains all the statistics extracted by the current and all former extractors. Consider an example of a control center with IP 1.2.3.4 sending periodic Modbus *read coils* requests to a substation with IP 4.3.2.1. The control center sends the request once per minute and tries to read coil 1 from device with UID of 2. After the request packet header goes through the OLAF collector, the extractors extract the *sender* of 1.2.3.4, the *receiver* of 4.3.2.1, the *protocol* of Modbus, the *UID* of 2, the *function* of READ_COILS, and *target* of 1. Note that if new operations in new protocols need to be supported, only the last two extractors need to be updated. All the other parts can remain exactly the same, which provides significant extensibility to the analyzer.

A useful feature of our analyzer is that users can easily scale the number of levels of statistics to collect. If the user is only interested in the flow-level information, the analyzer can be configured into a general network analyzer by only collecting the upper two levels of statistics. This can largely speed up the collector and make the analyzer more efficient, since the packet needs to go through the first extractor only instead of all three in this case. If operation-level statistics is required, the analyzer collects all 6 levels of statistics and provides the capabilities of device status analysis and network traffic analysis.

3.4.2 Statistics Aggregator

The Statistics Aggregator aggregates the information of each packet and constructs a tree structure $Tree_{new}$ to store the aggregated statistics. Each tree structure $Tree_{new}$ corresponds to statistics aggregated over certain period of time T_p and each node corresponds to statistics of a specific kind of packets. An example of the data structure is shown in Figure 3.3. Each node (leaf and internal) in the tree includes the following fields: (1)Info string IS, (2)Accumulated info string AIS, (3)Packet count PC, (4)Average byte AB, (5)Response ratio RR (*function* level node only), (6)Response delay RD (*function* level node only). IS is the value of the corresponding statistics level. For example, IS of *sender* level is its IP address and IS of *function* level is the function name. AIS of a node, on the other hand, is constructed by ISes of nodes on the path from the root to itself. And the node stores statistical data of the kind of packets represented by its AIS. For example, the node labelled "G_1" in Figure 3.3 has an AIS of "$S_1 - R_1 - P_1 - F_1 - G_1$" and therefore stands for packets

that sender S_1 sends to receiver R_1 using protocol P_1 with function F_1 performed on target G_1. The other four fields are data fields used to store statistical data of that kind of packets corresponding to the node during that period T_p. PC is the total number of the packets, while AB is the average size in bytes of the packets. RR and RD only exist for *function* level nodes and are the ratio of responded request functions and the delay of the response, respectively.

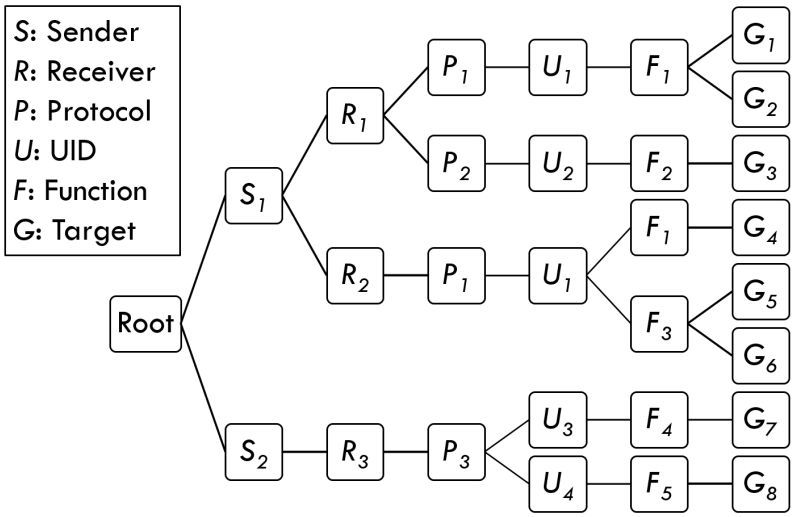


Figure 3.3: Statistics structure

In the workflow of the Statistic Aggregator, the *item_gen* event is fed into an item counter, which gets the information about the packet in the event and updates the corresponding nodes' date fields. There is also an aggregator which runs every period T_p, aggregating the results during that period as well as constructing the statistics structure. After the aggregator finishes the aggregation and construction, it triggers an *aggre_finish* event which includes the tree structure $Tree_{new}$ of that period. This event could be used for further analysis of the statistics of the network traffic such as anomaly detection which will be introduced in the following subsection.

3.4.3 Anomaly Detector

The Anomaly Detector is responsible for triggering alarms when anomalous traffic is seen in the network. There are mainly two approaches for intrusion detection: specification-base and anomaly-based. Although the anomaly-based approach has the ability to detect novel attacks, the difficulty of modelling the normal behaviour and the high false positive rate prevent it from widespread use. However, it is shown that the network traffic in SCADA

systems shows much more regularity than traffic in general purpose network [38]. Therefore, the anomaly-based approach is ideal for intrusion detection in Smart Grid SCADA networks. Our current anomaly detector uses the anomaly-based approach. Specifically, it utilizes a threshold-based algorithm, named *Normal Tree*. The core idea of the algorithm is constructing a 'normal' tree $Tree_k$ which represents the normal traffic and using it as a baseline. The next tree, constructed by the Statistic Aggregator, $Tree_{new}$ is then compared to the baseline to detect any potential anomaly.

The algorithm has two phases: initialization phase and anomaly detection phase. In the trusted initialization phase, which is the first k periods with total length of kT_p, the algorithm just merges the k trees and constructs the normal tree $Tree_k$. The structure of the normal tree $Tree_k$ is similar to $Tree_{new}$ except that we store a mean value μ and a standard deviation value σ for each statistic field (PC, AB, RR, RD) in each node. So each node has IS, AIS, (μ_{PC}, σ_{PC}), (μ_{AB}, σ_{AB}) (and additionally (μ_{RR}, σ_{RR}), (μ_{RD}, σ_{RD}) for *function* level nodes).

Algorithm 3.1 Normal Tree Algorithm

procedure AnomalyDetect($Tree_{new}, Tree_k, \theta$)
  Traverse $Tree_{new}$ in pre-order.
  for each node N in $Tree_{new}$: **do**
    if N also exists in $Tree_k$ **then**
      Use Equation 3.1 to calculate AS for each data field of N and compare them with θ. Continue to traverse N's children.
    else if N does not exist in $Tree_k$ **then**
      Assign $AS = 1$ to N instead of each data field and compare it with θ. Stop traversing N's children.
    end if
  end for
  Traverse $Tree_k$ in pre-order.
  for each node N in $Tree_k$: **do**
    if N does not exist in $Tree_{new}$ **then**
      Create a dummy node N with all data fields set to zero. Then use Equation 3.1 to calculate AS for each data field and compare them with θ. Stop traversing N's children.
    end if
  end for
end procedure

In the anomaly detection phase, we compare $Tree_{new}$ with $Tree_k$. One node N in $Tree_{new}$ is considered to be the "same" with another in $Tree_k$ if they have the same AIS. To compare the two same nodes in two trees, we assign anomaly scores to each data field of them. Suppose

the field in $Tree_{new}$ has a value of X and the corresponding field in $Tree_k$ has value μ and σ. Utilizing the Chebyshev's inequality, we define anomaly score AS as follows:

$$AS(X, \mu, \sigma) = \begin{cases} 1 - \frac{\sigma^2}{|X-\mu|^2} & \text{if } |X - \mu| > \sigma \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

The anomaly score is in the range $[0, 1]$ and a higher score represents more abnormal behavior. The algorithm then compares the score with a predefined threshold θ and decides whether to trigger an alarm or not. The anomaly detection phase of the algorithm is shown in Algorithm 3.1.

Consider our previous example of periodic *read coils* requests. Suppose this is the only traffic in the network and the substation never responds. We choose $T_p = 10min$, $k = 10$ and $\theta = 0.5$. The normal tree $Tree_k$ is represented by $Root_1 - S_1 - R_1 - P_1 - U_1 - F_1 - G_1$. All nodes have data fields $(\mu_{PC}, \theta_{PC}) = (10, 0)$, $(\mu_{AB}, \theta_{AB}) = (64, 0)$ and F_1 has additional data field $(\mu_{RR}, \theta_{RR}) = (0, 0)$. Now suppose the control center starts to send write coil requests instead of read coils requests to the same device and target at the same frequency. The next $Tree_{new}$ is represented by $Root_2 - S_2 - R_2 - P_2 - U_2 - F_2 - G_2$. The algorithm traverses $Tree_k$ and $Tree_{new}$ simultaneously in pre-order. It first finds that $Root_1$ and $Root_2$ have the equal AIS and therefore are same nodes. The calculated AS for both PC and AB are zeros, which are less than $\theta = 0.5$. Hence the algorithm does not trigger any alarm and continues to compare their children. Similarly, the algorithm compares S_1 with S_2, R_1 with R_2, P_1 with P_2, U_1 with U_2 in order and triggers no alarm. Then the algorithm finds that no node in $Tree_k$ has the same AIS with that of F_2, so it assigns $AS = 1$ to F_2. Since $AS = 1 > 0.5 = \theta$, it triggers an alarm. The algorithm also finds that no node in $Tree_{new}$ has the same AIS with that of F_1, so it creates a dummy node F_0 with all data fields set to zero and compares F_0 with F_1. $AS_{PC} = AS_{AB} = 1 > 0.5 = \theta$, hence the algorithm triggers two alarms.

The most criticized issue with the anomaly-based approach is the high false positive rate. There are three parameters in our analyzer that we can increase to reduce false positives: period time T_p, number of training periods k, and the anomaly score threshold θ. But they need to be carefully tuned since there are trade-offs and restrictions. Because we only detect anomalies at the end of each period, the period time T_p decides the maximum anomaly detection delay! We need to keep it small to provide prompt detection! A longer training phase has a larger training cost, and makes it harder to keep the training set clean. Increasing the threshold will make the detection algorithm less sensitive, which might decrease the detection rate. OLAF also has a feedback loop between the anomaly detector and the power

grid operator. If the operator finds one alarm to be false positive, he or she can give feedback to the detector so that the detector can adapt to avoid the same mistake. Our current naive feedback mechanism just increases the standard deviation in the corresponding data field in the normal tree. More sophisticated methods will be explored in the future work.

3.4.4 Pattern-based Identity Recognizer

The objective of the Pattern-based Identity Recognizer is to identify the device type of the traffic source and/or destination by monitoring certain request and response patterns in the traffic statistics.

The recognizer consists of a request-response coupler and a recognition rule matcher. The request-response coupler analyzes the packet statistics in the *item_gen* event. Based on the statistics, it couples each pair of request and response and constructs a variable which consists of the protocol and both the request and response function codes. For each pair of request and response, this variable is checked by the recognition rule matcher to see whether it matches the function pairs given by the recognition rules or not. If a match is found, the identities of the requester and/or the responder are also given by the corresponding matched rule and output by the matcher.

3.5 PERFORMANCE EVALUATION

In this section, we present the evaluation of OLAF in terms of time overhead and detection ability. We use a network analysis framework, called Bro [39], to implement our analyzer. All experiments run on a simulated Modbus trace set. The trace set was generated by a traffic simulator created by domain expert from Information Trust Institute at Illinois. The trace set includes a baseline traffic flow and some injected anomalies. In the baseline traffic, one Modbus master (control center) sends periodic operations to 10 Modbus slaves (field controllers). The valid operations in the baseline traffic and the frequencies of them are shown in Table 3.2.

3.5.1 Time Overhead Evaluation

Since our analyzer is inline to all traffic going through, we need to evaluate its processing time overhead to make sure it can handle real time traffic. We run OLAF on a 64-bit Ubuntu machine with 8 Intel i7-2600 3.40GHZ CPUs and 3.7GiB memory. In each of the following

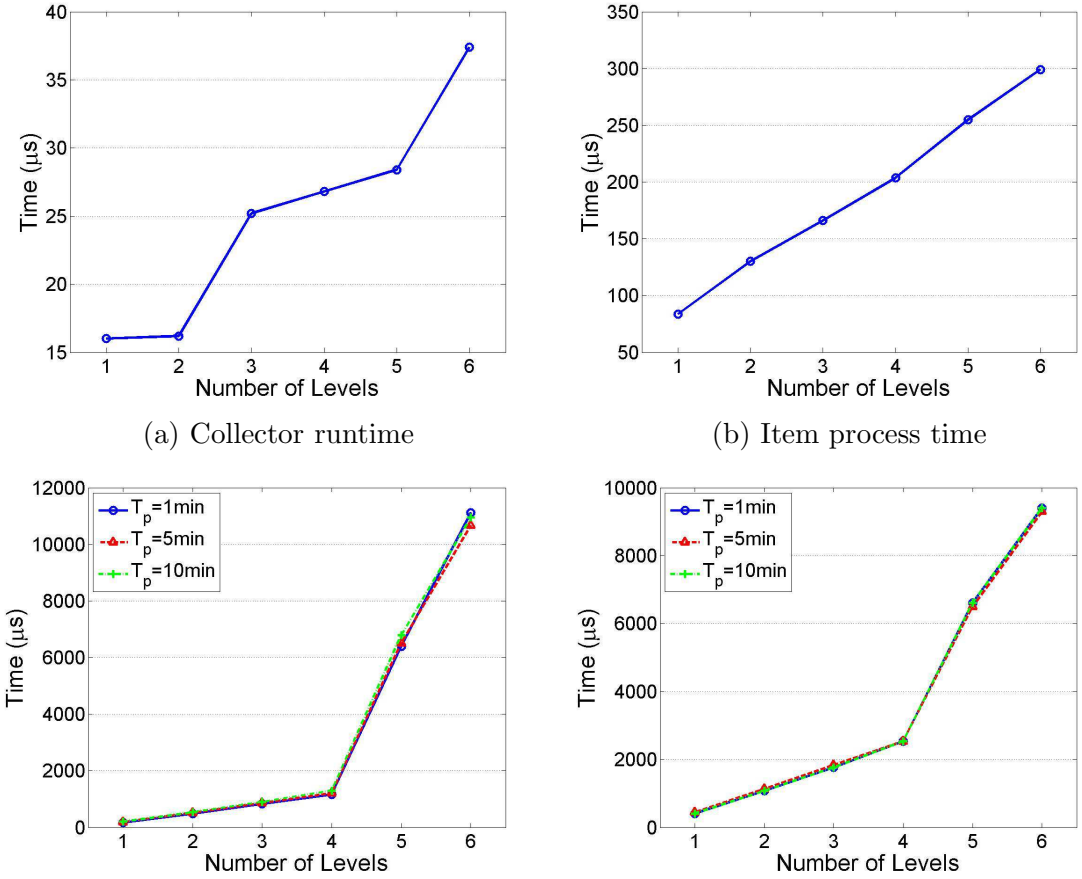
Operation	Function code	Frequency
Read Coils	01	Every 5 seconds
Read Discrete Inputs	02	Every 5 seconds
Read Holding Registers	03	Every 5 seconds
Read Input Registers	04	Every 5 seconds
Write Single Coil	05	Every 30 seconds
Write Single Register	06	Every 30 seconds
Write Multiple Coils	15	Every 5 minutes
Write Multiple Registers	16	Every hour

Table 3.2: Frequencies of valid operations in the baseline traffic

experiments, we run the corresponding modules on the same trace set for 5 rounds and take the average.

For the Statistics Collector, we measure the processing time of each individual packet and define it to be the total runtime of the collector. For the Traffic Statistics Counter, we are interested in the item process time and the aggregation time. Item process time is the time for the item counter to process the *item_gen* event and update the corresponding date fields and is measured per *item_seen* event. Aggregation time, which is measured per aggregation period, is the time for the aggregator to aggregate the data fields and construct the current tree structure. For the Anomaly Detector, the time to run the Normal Tree algorithm for one period is denoted by the anomaly detection time. Note that the total real-time processing time of each packet is the sum of collector runtime and item process time. And since the aggregation and anomaly detection is only done once for each aggregation period, they are not subject to the real-time overhead requirement.

Figure 3.4 shows the above four kinds of time overhead with different number of levels and aggregation period time. We can see that even if we collect statistics from all 6 levels, the total real-time processing time of each packet is still below 350 μs. And this is short enough for the packets to be processed in communication line speed. Moreover, T_p does not affect the aggregation time and anomaly detection time much since the traffic follows certain patterns. Most importantly, reducing the number of levels has a significant effect on the decrease of the time overhead of different modules. Therefore, always using the least necessary number of levels can save non-negligible amount of time and produces the highest processing speed. In this way, efficiency can be achieved by the analyzer.



(a) Collector runtime

(b) Item process time

(c) Aggregation time

(d) Anomaly detection time

Figure 3.4: Time overhead with different levels and aggregation period

3.5.2 Detection Ability Evaluation

To evaluate OLAF's anomaly detection ability, we inject anomalies in different levels to the baseline traffic. The injected anomalies are listed in Table 3.3. We fix the period time T_p to be 10 minutes and the anomaly detection phase to be 6 hours, and change the training phase length k and anomaly score threshold θ. We also evaluate the analyzer in both feedback off and on situations. OLAF is able to detect all nine anomalies in all cases, which means it never miss any anomaly. And we define ratio of number of false alarms to number of total node checked to be false alarm rate. The false alarm rate in different cases are listed in Table 3.4. We can see that increasing training time and anomaly score threshold both reduce false positives. And the feedback loop also has a huge effect on decreasing the false positive rate.

Level	Injected anomaly
Flow level	Add one new Modbus master to send a set of operations to one Modbus slave
	Drop one Modbus slave
	Send a bunch of ICMP packets to one Modbus slave
	Increase the response delay of one Modbus slave from 30ms to 200ms
	Increase the packet drop rate of one Modbus slave from 0 to 30%
Operation level	Stop sending some operations to one Modbus slave
	Send several new operations to one Modbus slave
	Change the sending frequency of some operations for one Modbus slave
	Change the targets of one operation for one Modbus slave

Table 3.3: Injected anomalies

Training Time		4 hours		8 hours	
Feedback		Off	On	Off	On
Threshold θ	0.9	4.16%	1.07%	3.01%	0.86%
	0.99	2.35%	0.62%	0.10%	0.06%
	0.999	2.26%	0.56%	0.05%	0.03%

Table 3.4: False alarm rate with different parameters

3.6 CONCLUSION

In this chapter, we present an extensible and efficient operation-level traffic analyzer framework, called OLAF, for Smart Grid SCADA networks to provide network traffic analysis and device status analyses. By collecting, aggregating and analyzing statistics in both flow level and operation level on the communication within the internal network, OLAF increases the situation awareness and security of the control system. Our results are strongly encouraging to place the extensible and efficient analyzer in Smart Grid SCADA networks.

CHAPTER 4: ISAAC: INTELLIGENT SYNCHROPHASOR DATA REAL-TIME COMPRESSION FRAMEWORK FOR WAMS

4.1 INTRODUCTION

Recently in Smart Grid, Wide-Area Monitoring Systems (WAMS) are becoming more and more widely accepted because of their ability to monitor, protect and control power systems over large areas in real time. WAMS's capability to support real-time decision-making applications is based on the high reporting rates and precise time synchronization provided by the new data acquisition technology of phasor measurement. Allowed by the emerging and development of Phasor Measurement Units (PMUs), frequency, current, and voltage can be measured at a rate of 30 Hz or higher, much faster than in the conventional SCADA systems, where samples are taken only every few seconds. The generated measurements are called synchrophasors, namely synchronized phasors, since they contain both magnitudes and phase angles, and are precisely time-synchronized by the Global Positioning System (GPS) technology. The synchrophasors generated by PMUs over wide-area power systems can serve as snapshots of the system status and can be further utilized for real-time wide-area monitoring, protection, and control. For instance, PMU data can aid or gradually replace the state estimation process which is a key function in supervisory control of power grids, since the accurate status information of the grid can be directly acquired from the real-time synchrophasors.

Since PMUs have very high sampling rates and usually multiple data channels, the volume of measurements collected is huge. 100 PMUs of 20 measurements, each running at 30 Hz, will generate over 50 GB of data per day [40]. 3500 data channels of 34 PMUs running at 100 Hz in Southwest China produce over 120 GB of data per day [41]. As the scale of power systems and WAMS grows, the number of PMUs is also growing rapidly to provide finer-grained global state of the more volatile power systems. For instance, the deployment of PMUs in North America has largely increased from only 200 research-grade PMUs in 2009 to almost 1700 production-grade PMUs in 2014 [42]. Besides the number of PMUs, the number of measurements at each PMU is also growing as more grid parameters get included for monitoring, from several synchrophasors to tens of them. Due to the higher sampling rate of modern PMUs, the increase in the number of PMUs and measurements per PMU, we can surely expect a multi-fold expansion in the already large volumes of synchrophasor data in WAMS.

The synchrophasor data generated by PMUs need to be transmitted in the underlying communication systems in real time and stored in control centers for archive purpose (his-

torian). The huge and ever-increasing volume of synchrophasor data introduces tremendous storage and bandwidth capacity requirement for WAMS. Therefore, it is necessary to use data compression techniques to lighten the heavy burden on the storage and communication systems. Many works of power data compression focus on the compression for storage or offline bulk transmission [43, 44, 45, 46, 47, 48, 49]. However, we argue that online data compression for real-time transmission should be addressed as much, if not more, than offline compression. If not handled carefully, the huge data volume in the communication system could result in frequent and severe congestion. The WAMS applications and even the situational awareness of the entire system could suffer a lot from the extremely long delays or high packet loss rates that follow the congestion.

The two main challenges for designing real-time compression frameworks in WAMS are as follows:

- *Delay*: The data should be compressed in a real-time manner. In other words, the delay matters. Most existing compression techniques use large sampling windows to achieve better compression performance, which is a luxury that real-time compression techniques cannot afford.

- *Disturbance*: The delay and accuracy requirements of data during disturbance[1] are different from those in normal status. Thus it is necessary to treat disturbance data and normal data differently in compression, which requires the early detection of disturbances to be incorporated into the compression framework.

In this work, we propose an intelligent synchrophasor data real-time compression framework named ISAAC to be deployed at the edge of WAMS. Combining the Principal Component Analysis (PCA) and Discrete Cosine Transform (DCT), ISAAC has the capability of largely improving the efficiency of communication and storage systems via data compression while maintaining strong data fidelity. A disturbance detector allows ISAAC to differentiate normal and disturbance data and process them in different ways. Two core techniques, which are the transformation matrix reuse in PCA and the self-adapt principal component number selection, allows ISAAC to achieve a good compression ratio (CR) without introducing an impractical delay.

The remainder of this chapter is organized as follows: Section 4.2 reviews the related work. Section 4.3 provides some background and explains the two compression algorithms used by our approach. Section 4.4 describes the design of ISAAC. Section 4.5 shows the performance evaluation of time, CR, and accuracy and Section 4.6 concludes the chapter.

[1]Disturbances here means transients in measurements that may be caused by misoperations, faults, topology changes, load and source dynamics.

4.2 RELATED WORK

Data compression techniques can be categorized into lossless compression and lossy compression in general. There are many works [43, 44, 45] focusing on lossless compression of power quality data, smart meter data, and PMU data. Our work, on the contrary, focuses on lossy compression, since lossy compression has a potential to achieve a better CR compared with lossless compression and it is acceptable as long as parameters of compression algorithms are selected carefully to maintain information loss within required bound.

Among all the works using lossy compression techniques for PMU data, most of them [46, 47, 48, 49] perform compression for storage or offline bulk transmission purpose, in which cases the delay of the compressed data is not a concern. Therefore, they are able to use a sampling window with the length of 10 seconds or longer to gather enough data and compress the entire data block to achieve better CRs. For sampling window based approaches, the earliest set of measurements in each window needs to wait for the entire window to be filled before it can be compressed and sent. Hence, a window size of 10 seconds means a delay of more than 10 seconds for the earliest set of measurements in each data block, which is far beyond the delay constraints of most of the real-time WAMS applications [50, 51]. As a result, compression techniques for PMU data storage or offline transmission are not directly applicable to real-time PMU data compression purpose.

A semantics-aware real-time data transmission reduction method is proposed in [52]. Grid applications consuming PMU data are modelled as continuous threshold queries and relevant data are delivered only when the threshold condition is broken. The drawback of this approach is not all applications can be modeled in their way and absence of detailed data during normal status will definitely impair the system's ability to conduct monitoring and detailed analysis of the data.

Another real-time data compression technique is presented in [41], combining exception compression with swing door trending compression. Each sequence of measurements of each PMU is compressed separately by only keeping the data with essential and effective information. The problem with this approach is that the correlation between multiple sequences of measurements in multiple PMUs is not utilized. And since this technique needs to be installed on each PMU, the deployment cost is high considering the large quantity of PMUs.

4.3 PRELIMINARIES

In this section, we first introduce the generic WAMS architecture and the phasor data concentration. Then we introduce two compression techniques we use.

4.3.1 WAMS Architecture

A generic architecture of WAMS consists of four main components: (1) PMU, (2) Phasor Data Concentrator (PDC), (3) WAMS Applications, (4) underlying Communication Network to connect the above three [53]. A simplified, multi-layered architecture of WAMS is shown in Figure 4.1. In Layer 1, the PMUs are installed in power system substations to measure the connected bus bars or power lines. The synchrophasor measurements from the PMUs are then transmitted via Local Area Networks (LAN) or Wide Area Networks (WAN) to Layer 2, where they are concentrated and sorted by the PDCs. After that, the time-aligned measurements are forwarded via WAN to the control center in Layer 3 and usually further concentrated by the PDC there before finally consumed by the WAMS applications.

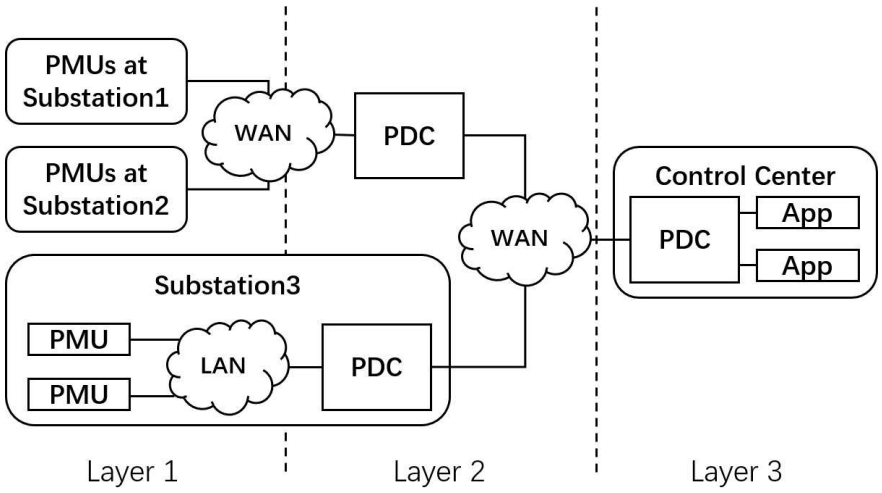


Figure 4.1: WAMS architecture

Our real-time compression technique, ISAAC, resides in the PDCs in Layer 2. We further divide the architecture into two scenarios: (1) LAN-WAN scenario, where the Layer-2 PDC locates in the substation and connects to the PMUs via LAN; (2) WAN-WAN scenario, where the Layer-2 PDC collects measurements from multiple substations and connects to the PMUs via WAN.

4.3.2 Phasor Data Concentration

As the key component of the WAMS architecture, PDC's core function is to combine synchrophasor measurements from more than one PMUs into a single time-synchronized data stream for further processing [54]. More specifically, it collects and sorts measurements from all connected PMUs according to their timestamps. Measurements with the same

timestamp are encapsulated into one packet and forwarded to the control center. Since not all measurements arrive at the same time, PDC needs to wait and eventually timeout to mitigate the delay. The entire process is called time alignment and can be categorized into absolute-time-based and relative-time-based time alignment [54]. For real-time monitoring, protection and control applications, time alignment to absolute time reference is preferred since it can provide better delay guarantees. Therefore, we will only consider and introduce time alignment to absolute time reference here.

An example of phasor data concentration with time alignment to absolute time is shown in Figure 4.2. At a certain rate (e.g., 60Hz), measurements with timestamps are produced by synchronized PMUs. The countdown to the timeout starts at the time specified by each timestamp. As it is shown in the figure, there are two potential scenarios: (1) All measurements with timestamp T1 arrive before the timeout and the complete data set is forwarded before the timeout; (2) Not all measurement with timestamp T2 arrive before the timeout and the incomplete data set without measurement from PMU_3 is forwarded at the end of the timeout. Note that the processing time of PDC is omitted in this example for simplicity reason.

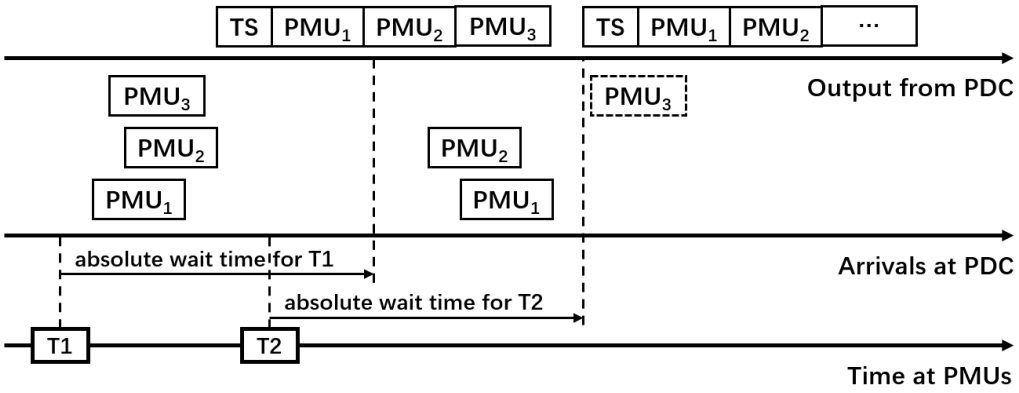


Figure 4.2: Example of phasor data concentration with time alignment to absolute time [55]

Phasor data concentration is only the core function of the PDC. Since the PDC is an edge device that phasor measurements reach before they are further forwarded to the control center, more and more functions are being placed at the PDC including data handling, processing, and storage [54]. And it is also a perfect location to deploy our intelligent synchrophasor data real-time compression framework.

4.3.3 Principal Component Analysis

As a commonly used linear dimensionality reduction technique, PCA [56] is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of linearly uncorrelated variables called principal components (PCs).

Consider an $m \times n$ data matrix $\mathbf{X}$ containing n set of samples from m PMUs[2] expressed as

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}. \tag{4.1}$$

x_{ij} represents the jth measurement from the ith PMU. The PCA method starts by calculating the covariance matrix as $\mathbf{C} = \mathbf{X}\mathbf{X}^T \in \mathbb{R}^{m \times m}$. Since $\mathbf{C}$ is a square symmetric matrix, it can be orthogonally (orthonormally) diagonalized as

$$\mathbf{C} = \mathbf{E}\mathbf{D}\mathbf{E}^T, \tag{4.2}$$

where $\mathbf{E}$ is an $m \times m$ orthonormal matrix whose columns are the eigenvectors of $\mathbf{C}$, namely PCs, and $\mathbf{D}$ is an $m \times m$ diagonal matrix with the corresponding eigenvalues as the diagonal entries. The eigenvalues can also represent the variance explained by each PC and are sorted in descending order. PCA performs dimensionality reduction by preserving only a subset of PCs which explain most of the variance of the original data. Assume the first r out of the m PCs are selected. Let $\mathbf{E}(r)$ represent the left most r columns of $\mathbf{E}$. The transformation matrix is selected as $\mathbf{P} = \mathbf{E}(r)^T \in \mathbb{R}^{r \times m}$ and the dimension reduced matrix is expressed as $\mathbf{Y} = \mathbf{P}\mathbf{X} \in \mathbb{R}^{r \times n}$. Let λ_i represent the eigenvalue (variance), associated with the ith PC, the total variance explained by $\mathbf{Y}$ is defined as

$$\Gamma(r) = \frac{\sum_{i=1}^{r} \lambda_i}{\sum_{i=1}^{m} \lambda_i}. \tag{4.3}$$

Usually, we want to select r s.t. $\Gamma(r)$ is greater or equal to a variance threshold γ.

Although the core idea of dimensionality reduction by PCA is the same as above, there are various implementations of the method. In this work, we use two implementations: The first one (we name it PCA-D) utilizes the *sklearn.decomposition.PCA* [57] implementa-

[2]There are usually multiple measurements from each PMU. So the actual number of rows is larger than the number of PMUs. But we assume m rows here for simplicity.

tion and cannot work with sparse matrix; the second one (we name it PCA-S) utilizes the *sklearn.decomposition.TruncatedSVD* [57] implementation and can preserve the sparsity of the matrix.

4.3.4 Discrete Cosine Transform

DCT [58] transforms a sequence of data points of length n to a domain of n cosine basis functions. The transformed data are the coefficients of the basis functions. Some of the coefficients have small magnitudes and thus can be discarded without sacrificing much accuracy. We use DCT implemented in *scipy.fftpack.dct* [59] to further compress each row of the dimension reduced matrix $\mathbf{Y}$ in the previous section. Similar to the previous section, let c_i represent the coefficient with the ith largest magnitude. If l out of all the n coefficients are kept, the cumulative energy kept can be expressed as:

$$E(l) = \frac{\sum_{i=1}^{l} c_i}{\sum_{i=1}^{n} c_i} \tag{4.4}$$

Similar to $\Gamma(r)$, $E(l)$ is used to select proper l value. In this work, we select the smallest l that satisfies $E(l) > 0.8$.

4.4 DESIGN OVERVIEW

In this section, we present an overview of the design of ISAAC. The workflow of ISAAC is described in Figure 4.3. There are four main components utilized in ISAAC: (1) *Buffer* which buffers all the measurements have not been sent, (2) *Time Alignment Component* which time-aligns the most recent set of measurements with an absolute timeout, (3) *Disturbance Detector* which decides whether there are disturbances happening, (4) *Compressor* which compresses the input matrix based on PCA and DCT. Among the four components, the buffer and the time alignment component are the built-in functions of PDCs. The disturbance detector and the compressor, on the other hand, are the core parts of ISAAC and are the focus of this section.

In Figure 4.3, $\vec{x}_i$ is a column vector containing all the received measurements from all PMUs corresponding to time index i. The buffer serves as the sampling window and buffers all the received stream measurements with time index k or larger. $\vec{x}_k$ represents the earliest unsent data set and $\vec{x}_n$ represents the earliest unprocessed data set. Periodically, the time alignment component aligns the measurements in the buffer based on their timestamps

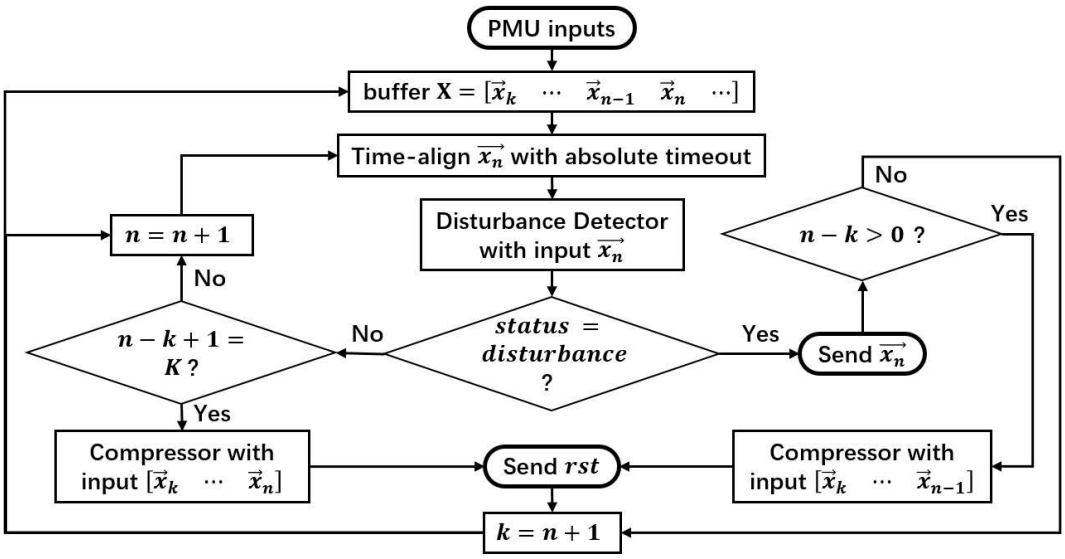


Figure 4.3: Workflow of ISAAC

and forwards $\vec{x}_n$ to the disturbance detector[3]. The disturbance detector processes $\vec{x}_n$ and outputs the estimated status (normal or disturbance) of the system. In normal status, ISAAC further compares the current window size from k to n with the maximum sampling window size K. If $n - k + 1 = K$, all measurements buffered from k to n, i.e. matrix $[\vec{x}_k, \cdots, \vec{x}_n]$, are forwarded to the compressor and compressed. Otherwise, ISAAC waits for more measurements (longer sampling window) to compress. In disturbance status, $\vec{x}_n$ is directly sent without compression. If there are buffered measurements besides $\vec{x}_n$, i.e. matrix $[\vec{x}_k, \cdots, \vec{x}_{n-1}]$ is not empty, then they are forwarded to the compressor for compression. If the compressor is called, no matter in which status, the compressed results are sent. k is updated to $n + 1$ as long as something is sent, which means the measurements in the buffer before but not including time index $n + 1$ are cleared. And n is always increased by one after each period.

There are five kinds of measurements we consider in this work: *Frequency* (f), *Voltage Magnitude* (vm), *Voltage Angle* (va), *Current Magnitude* (im), and *Current Angle* (ia). The following two subsections describe the disturbance detector and the compressor in more detail.

[3]There could be missing data points in $\vec{x}_n$ and in the input matrix of the compressor. We fill in each missing data point by its current mean and record it by a boolean. We omit this process in the workflow for simplicity.

4.4.1 Disturbance Detector

The main purpose of the disturbance detector is to differentiate the normal status and the disturbance status of the system. Intuitively, a higher delay and lower accuracy are tolerable in normal status, whereas measurements should be collected as soon as possible during disturbances and higher fidelity is necessary to preserve the important information in the measurements. Since the requirements for delay and accuracy are different in the two status, it is worth differentiating them and handle them in different ways. Our disturbance detector implements a modified version of a relatively simple statistical change detection algorithm [48] for computation power and delay consideration.

According to PRC-002-2 by NERC [60], the recommended disturbance triggering criteria include: (1) frequency < 59.75 Hz or > 61 Hz, (2) rate of change of frequency < 0.03125 Hz/s or > 0.125 Hz/s, 3) Undervoltage trigger set no lower than 85% of the normal operating voltage for a duration of 5 seconds. However, as it is pointed out in [61, 48], these values are too conservative to detect all potential disturbances and preserve all critical information. Similar to [61], we select stricter triggering criteria as percentage deviation of $\theta_{vm} = 1\%$ for voltage and $\theta_f = 0.1\%$ for frequency. These values could be tuned if necessary without affecting the algorithm itself.

Since the disturbance triggering criteria are based on frequency and voltage magnitude only, the disturbance detector only processes those two kinds of measurements. The workflow is shown in Figure 4.4. First of all, the current status is checked and there could be two scenarios: (1) The system is in the normal status; (2) The system is in the disturbance status. In the first scenario, for each $i \in 1 \dots m$, the deviation $\delta_i(n) = |x_{in} - \mu_i(n)|$ is calculated, where $\mu_i(n) = \frac{1}{K} \sum_{j=n-K}^{n-1} x_{ij}$ is the current mean. If all the percentage deviations are smaller than the triggering threshold, namely $\delta_i(n)/\mu_i(n) < \theta$ for all $i \in 1 \dots m$, no disturbance is detected and the normal status remains. Otherwise, status is changed to disturbance and the disturbance count is set to 1. In the second scenario, we assume a disturbance will last for at least 3 periods. So if the disturbance count is smaller than 3, the disturbance status remains and the count is increased by 1. Otherwise, for each $i \in 1 \dots m$, a special standard deviation of the most recent 3 periods are calculated as $\sigma_i(n) = \sqrt{\frac{1}{3} \sum_{j=n-2}^{n} (x_{ij} - \mu_i(n))^2}$. If all the percentage standard deviations are smaller than the triggering threshold, namely $\sigma_i(n)/\mu_i(n) < \theta$ for all $i \in 1 \dots m$, the disturbance is considered over and the status is changed back to normal. Finally, the detector outputs the current status in all cases.

Examples of using the disturbance detector for both frequency and voltage are shown in Figure 4.5. Blue lines represent normal status and red lines represent disturbance status. We can see that for both measurements, the disturbances can be detected quickly and

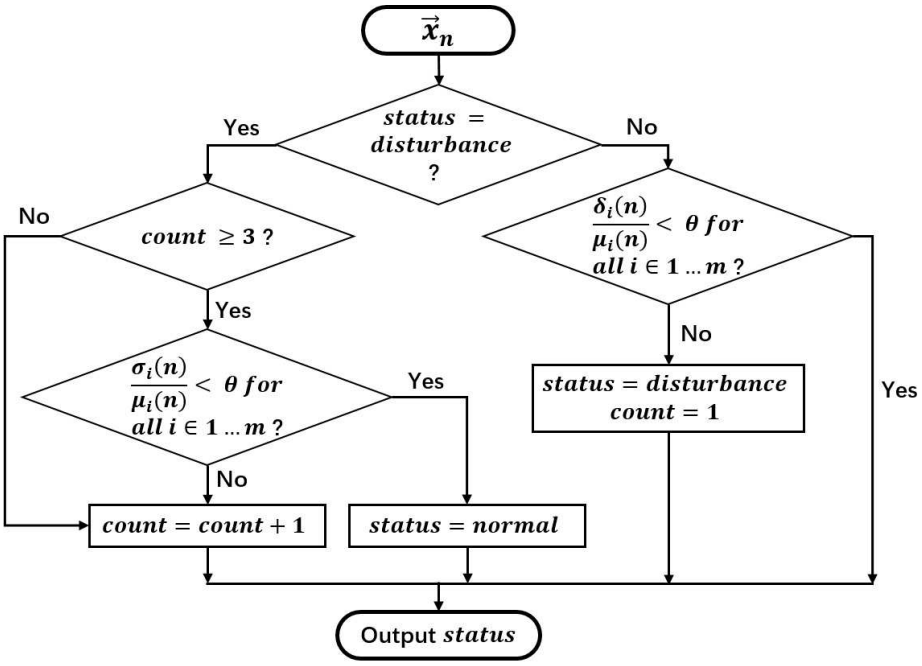
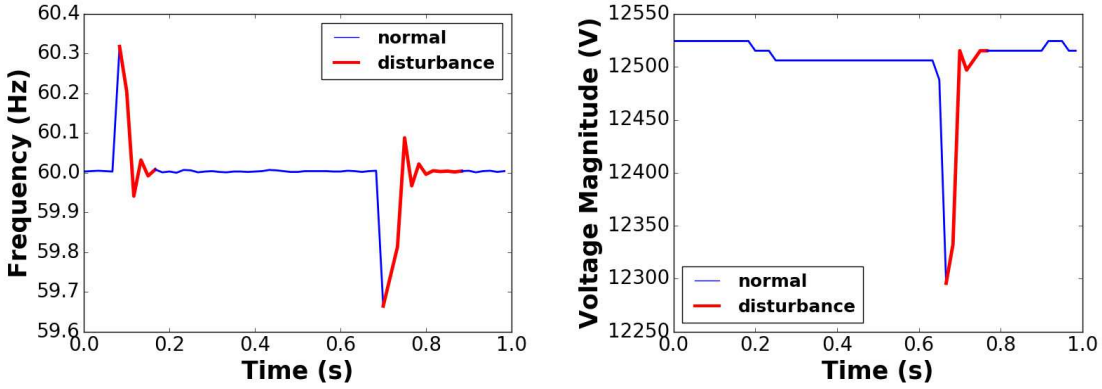


Figure 4.4: Workflow of the disturbance detector

the normal status is retained soon after the disturbances end. Note that the status here describes the entire system, therefore it is shared among all the measurements. So as long as one measurement (f or vm) in one of the PMUs contains disturbances, all the five kinds of measurements from all PMUs for this PDC are considered in disturbance status.



5

(a) Frequency (b) Voltage

Figure 4.5: Example of disturbance detection

4.4.2 Compressor

The purpose of the compressor is to use a combination of PCA and DCT to compress the input matrix in an intelligent way in order to reduce the data volume to send while keeping the sampling window small and maintaining a certain accuracy for the reconstructed data. It mainly uses two techniques to achieve that: the transformation matrix reuse in PCA and the self-adapt PC number selection. The measurements of each kind are processed separately and in parallel. Hence, there are actually five instances of the compressor running simultaneously, each for one of the five kinds of measurements.

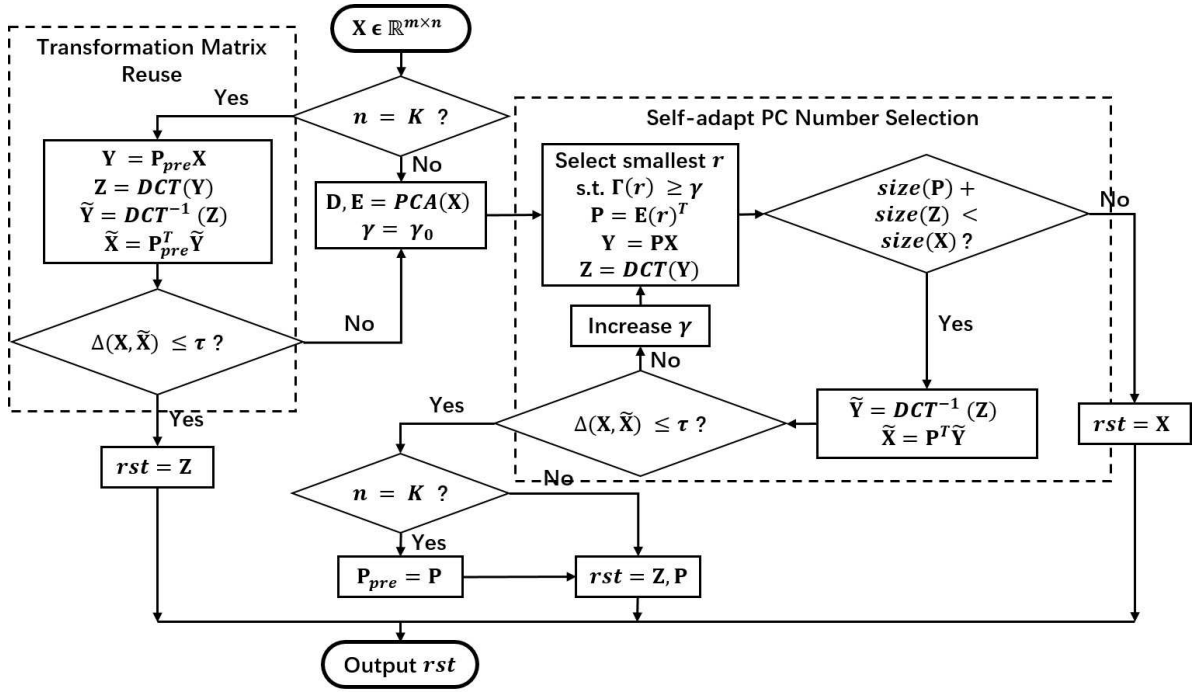


Figure 4.6: Workflow of the compressor

The workflow of the compressor is shown in Figure 4.6. Assume the input is an $m \times n$ matrix $\mathbf{X}$. The compressor first checks whether n is equal to the maximum window size K. If $n = K$, it means the system is in normal status and the matrix is of standard size $m \times K$. The transformation matrix reuse module is then called. If $n < K$, it means the system is in disturbance status and the self-adapt PC number selection module is called. These two modules are explained as follows:

- *Transformation Matrix Reuse*: The intuition behind this module is that temporally closed standard matrices could share very similar PCs, namely transformation matrix $\mathbf{P}$. We use $\mathbf{P}_{pre}$ to represent the latest transformation matrix calculated and sent to

the receiving side in previous iterations[4]. Whenever a new data block is ready to be compressed, instead of recalculating and resending each time a new transformation matrix, $\mathbf{P}_{pre}$ is tested for reuse. To be more specific, the module compresses $\mathbf{X}$ using $\mathbf{P}_{pre}$ followed by DCT. Then it reconstructs $\widetilde{\mathbf{X}}$ by inverse DCT and inverse PCA based on $\mathbf{P}_{pre}$. The reconstruction accuracy is evaluated by the maximum relative error defined as[5]:

$$\Delta(\mathbf{X}, \widetilde{\mathbf{X}}) = \max_{\substack{i=1\ldots m \\ j=1\ldots n}} \frac{|x_{ij} - \tilde{x}_{ij}|}{\xi_i} \tag{4.5}$$

where ξ_i equals the current mean defined in 4.4.1 for f, vm, im and equals to 360 for va and ia. If the maximum relative error is less or equal to the specified tolerance represented as τ[6], $\mathbf{P}_{pre}$ can be reused and the compressor only outputs the compressed data. Otherwise, $\mathbf{P}_{pre}$ cannot be reused and the self-adapt PC number selection module is called.

- *Self-adapt PC Number Selection*:

  While using PCA to perform dimensionality reduction, one needs to decide the number of PCs to keep, represented as r. The trade-off here is that decreasing r will decrease the size of the compressed data but increase the error of reconstructed data. The purpose of the self-adapt PC number selection module is to select the proper r to minimize the compressed data size while keeping the reconstructed error in a certain threshold τ. Equation 4.3 and a variance threshold γ is used to select r. The module works in an iterative way. The iteration starts after matrices $\mathbf{D}$ and $\mathbf{P}$ are calculated according to equation 4.2 and γ is initialized as γ_0. In each iteration, the smallest r that satisfies $\Gamma(r) \geq \gamma$ is selected, where $\Gamma(r)$ is defined in equation 4.3. The transformation matrix is then selected as $\mathbf{P} = \mathbf{E}(r)^T$. The compressed matrix $\mathbf{Z}$ is calculated by projecting $\mathbf{X}$ by $\mathbf{P}$ followed by DCT. If $size(\mathbf{P}) + size(\mathbf{Z}) < size(\mathbf{X})$, data size is reduced after compression and the iterations continues. Otherwise, no size reduction is gained after compression and the original matrix $\mathbf{X}$ is assigned to the result. After the size check, $\widetilde{\mathbf{X}}$ is reconstructed by inverse DCT and inverse PCA. The maximum relative error is calculated and compared with the tolerance. If $\Delta(\mathbf{X}, \widetilde{\mathbf{X}}) \leq \tau$, the current r satisfies the required reconstruction accuracy and the compressed matrix $\mathbf{Z}$ and transformation matrix $\mathbf{P}$ are assigned to the result. Otherwise, the variance threshold γ is increased

[4] $\mathbf{P}_{pre}$ is only updated when a new $\mathbf{P}$ is calculated and actually sent.

[5] This is slightly different from the standard definition of maximum relative error

[6] τ has different values for different measurement kinds and τ should always be smaller than θ to prevent reconstruction error from triggering disturbances.

and a new iteration begins. Note that the $\mathbf{P}$ in result are recorded by $\mathbf{P}_{pre}$ only when the compression succeeds and X is of standard size, namely $n = K$.

4.5 PERFORMANCE EVALUATION

In this section, we present the performance evaluation of ISAAC in terms of time, CR, and accuracy. The experiments run on a dataset consisting of field synchrophasor data collected from a microgrid at Illinois Institute of Technology (IIT) [62]. The dataset contains 18 hours of data (including disturbances) collected from 11 PMUs running at 60 Hz from 6 PM, 1/28/2014 to noon, 1/29/2014. The dataset includes 11 sequences of frequency measurements, 94 sequences of voltage synchrophasors, and 119 sequences of current synchrophasors.

The parameter values used in ISAAC are shown in Table 4.1. Note that $\tau_f < \theta_f$ and $\tau_{vm} < \theta_{vm}$, so that reconstruction error won't trigger disturbances. With the maximum sampling window size $K = 10^7$ and arrival rate of 60 samples/second, the sampling window of 10 samples has a length of 167 ms. All the parameter values are not fixed and can be tuned by users to satisfy their own needs. Of the two WAMS architecture scenarios mentioned in 4.3.1, we choose the WAN-WAN scenario for experiments since it is likely to have a longer communication delay and serves as a worse case for validation. We assume all the 11 PMUs are connected to one PDC via WAN and the PDC is connected with a control center via WAN. According to [63], the communication delay between a PMU and a PDC connected by WAN follows a bimodal distribution containing two normal distributions. We assume the communication delays between each pair of PMU and PDC and between PDC and control center are all i.i.d. random variables following the same bimodal distribution with $p = 0.5$, $\mu_1 = 10$ ms, $\sigma_1 = 1$ ms, $\mu_2 = 16$ ms, $\sigma_2 = 3$ ms, where p is the mixing factor. The absolute timeout value of the PDC is set to 25 ms.

Parameter	Value	Parameter	Value	Parameter	Value
K	10	τ_f	0.025%	τ_{im}	4%
θ_f	0.1%	τ_{vm}	0.2%	τ_{ia}	0.5%
θ_{vm}	1%	τ_{va}	0.5%	γ_0	0.5

Table 4.1: Default parameter values of ISAAC

The compression ratio is calculated by the raw data size divided by the compressed data size. And the accuracy of reconstructed data is measured in terms of the maximum percentage error (MPE) and the normalized root-mean-square error (NRMSE). Assume the original

[7] A larger K will result in a higher compression ratio but also a higher delay. Here we choose $K = 10$ for a trade-off.

data matrix is $\mathbf{X} \in \mathbb{R}^{m \times t}$ and the reconstructed matrix is $\widetilde{\mathbf{X}} \in \mathbb{R}^{m \times t}$. MPE and NRMSE can be calculated as

$$MPE = \max_{\substack{i=1\ldots m \\ j=1\ldots t}} \frac{|x_{ij} - \tilde{x}_{ij}|}{\mu_i} \times 100\% \tag{4.6}$$

$$NRMSE = \frac{1}{m} \sum_{i=1}^{m} \sqrt{\frac{\sum_{j=1}^{t} (x_{ij} - \widetilde{x}_{ij})^2}{t}} \Bigg/ \mu_i \ , \tag{4.7}$$

where $\mu_i = \sum_{j=1}^{t} x_{ij}/t$ for f, vm, im, and $\mu_i = 360$ for va and ia.

There are two implementations of ISAAC, namely PCA-D/DCT and PCA-S/DCT. The CR, MPE, NRMSE of each type of measurement for the two implementations are shown in Table 4.2. It can be seen that good CRs can be achieved while maintaining satisfactory reconstruction accuracy. The only exception is the current magnitude data, where it is hard to compress without introducing relatively large MPE. This is due to the extremely noisy current magnitude data we observe. In this case, it might not worth compressing current magnitude data to avoid sacrificing data fidelity.

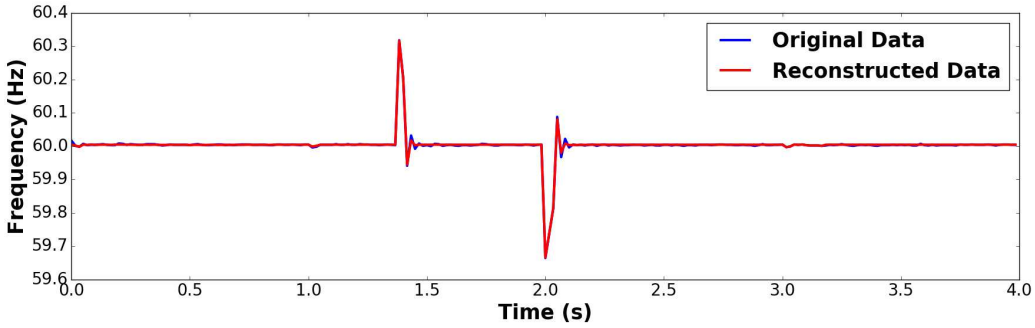
Measurement	PCA-D/DCT			PCA-S/DCT		
	CR	MPE (%)	NRMSE	CR	MPE (%)	NRMSE
f	9.11	0.025	4.12E-5	5.21	0.025	3.01E-5
vm	20.36	0.202	3.36E-4	19.86	0.202	3.31E-4
va	20.58	0.500	1.20E-3	21.45	0.500	1.05E-3
im	2.72	6.937	5.02E-3	1.31	6.937	5.05E-3
ia	3.52	0.500	5.96E-4	1.47	0.500	6.87E-4

Table 4.2: CR, MPE, and NRMSE of two implementations

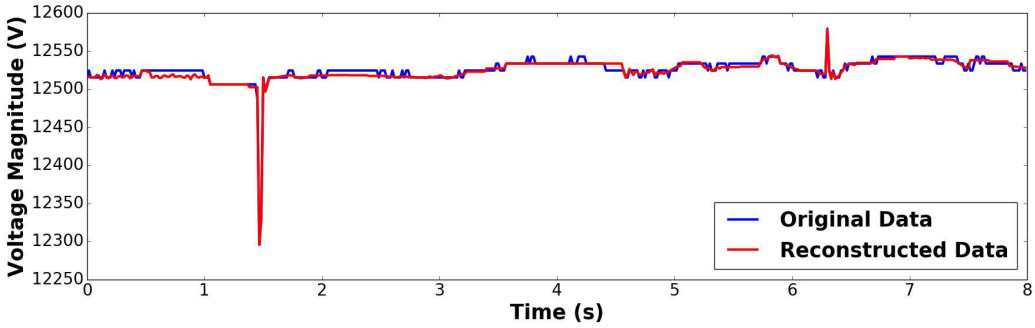
Figure 4.7 illustrates a comparison between original data (blue lines) and reconstructed data (red lines) for two sample frequency and voltage measurement sequences, based on the PCA-D/DCT implementation of ISAAC. As we can see from the figure, most noises are discarded while critical changes and disturbances are preserved. We also evaluate the performance of DCT alone on data compressed by PCA already. After compressing the data by PCA-D, DCT can further reduce the data size by 34.2% for f and 14.2% for vm.

Based on the accuracy requirements of the WAMS applications, τ can be changed to achieve different accuracy levels. With various τ_f, the CR, MPE, NRMSE of frequency measurements based on the PCA-D/DCT implementation of ISAAC is shown in Table 4.3[8]. We can see that the CR and reconstruction error both decrease as τ decreases. Therefore,

[8]These results are based on one hour of data in the IIT dataset.



(a) Frequency



(b) Voltage

Figure 4.7: Original and reconstructed data

usually the largest τ that satisfies the accuracy requirements should be selected to maximize the CR.

τ_f	CR	MPE (%)	NRMSE
0.001	18.31	0.1	1.52E-4
0.0005	14.27	0.05	6.42E-5
0.00025	8.37	0.025	4.20E-5
0.0001	3.44	0.01	2.37E-5
0.00005	1.79	0.005	1.32E-5

Table 4.3: CR, MPE, and NRMSE of frequency with various τ_f

To demonstrate ISAAC's ability to satisfy the real-time requirements of WAMS applications, we also evaluate and processing time of different components and the end-to-end data delay for both normal and disturbance status. The end-to-end delay is a sum of the communication delay between PMU and PDC, PDC processing delay, communication delay between PDC and control center, and the data reconstruction time. We run ISAAC on a Mac with an Intel Core i7 1.7GHz CPU and 8GB memory. The average processing time of

the disturbance detector, compressor and data reconstruction (done at the control center) for each period is shown in Table 4.4. It can be seen that the processing time is quite negligible and should be even shorter on real PDCs which are much more powerful devices than our Mac. The average and the maximum end-to-end delays of normal and disturbance status are shown in Table 4.5. We can see that the end-to-end delays in disturbance status are significantly smaller than delays in normal status, which is desirable by most of the WAMS applications. And a maximum delay of 201 ms satisfies the delay requirements of real-time WAMS applications [64, 65].

disturbance detector	0.12
compressor	4.75
reconstruction	0.46

Table 4.4: Avg. Processing Time (millisecond)

Status	Avg	Max
normal	107.1	201.4
disturbance	32.5	49.6

Table 4.5: End-to-End Data Delay (millisecond)

4.6 CONCLUSION

In this chapter, I present ISAAC, an intelligent synchrophasor data real-time compression framework for WAMS to be deployed in Layer 2 PDCs. Based a combination of PCA and DCT techniques, ISAAC is able to mitigate the burden on communication and storage systems laid by the huge synchrophasor data volume while satisfying the requirements of real-time WAMS applications. A disturbance detector is utilized to identify disturbance data and satisfy its stricter delay and accuracy requirements. The use of two techniques named transformation matrix reuse in PCA and self-adapt PC number selection enables ISAAC to achieve good compression ratios while maintaining satisfying delay and accuracy for the reconstructed data. The performance of ISAAC is validated by experiments based on real synchrophasor data.

CHAPTER 5: EDMAND: EDGE-BASED MULTI-LEVEL ANOMALY DETECTION FOR SCADA NETWORKS

5.1 INTRODUCTION

As stated in 1.1, SCADA systems in Smart Grid are subject to a wide range of serious threats in recent years and they could suffer from catastrophic consequences due to successful attacks. Well-known malicious cybersecurity incidents in SCADA systems include the Stuxnet worm attack [66] and the BlackEnergy malware [67]. These attacks exploited the vulnerabilities of SCADA systems and the situation is expected to deteriorate further for several reasons. First, the adoption of cutting-edge communication technologies contributes to the increasing complexity and interconnection of SCADA systems, which potentially provides greater opportunity for attacks from malicious sources. Since corporate intranets can be connected to the internet, SCADA systems connections with corporate intranets may expose their communication weakness to threats of broader aspects. Second, devices in SCADA systems are usually not built with cybersecurity in consideration and lack authentication or encryption mechanisms. To make things worse, the enabling of remote access to these devices via wireless technologies makes them easy to compromise. Third, most ICS protocols lack authentication features and provide no protection for the network traffic. The vulnerabilities of SCADA systems can be exploited from both outside by malicious attackers and inside by disgruntled employees. Besides deliberate attacks, inadvertent events such as natural disasters, device failures, and operator mistakes may also jeopardize SCADA systems due to those vulnerabilities. Therefore, developing techniques to target those vulnerabilities and provide security to SCADA systems is a pertinent topic of particular importance.

In general, two types of analysis are available to provide security for SCADA systems: host-based and network-based. We focus on network-based analysis which monitors and inspects network traffic due to its less intrusive nature. In [18], I propose a light-weighted operation-level traffic analyzer named OLAF to provide preliminary analysis of SCADA. That is not enough to guarantee situational awareness and a more thorough monitoring and analysis and required. Based on different analysis granularity, data in SCADA network traffic generally can be divided into three levels: transport level, operation level, and content level. Transport level data refers to statistics in IP headers and transport protocol headers. Operation level data refers to operation statistics in ICS protocols. Content level data refers to measurement statistics from field devices. Among all network-based security analysis approaches for SCADA systems, most existing solutions only focus on monitoring and event detection of one or two levels of data, which is not enough to detect and reason about attacks

in all three levels. Also, data in each level has its own characteristics, which requires distinct methods to deal with. In this work, we develop an edge-based multi-level anomaly detection framework for SCADA networks, named EDMAND. EDMAND is located inside the remote substations, which are the edges of the SCADA network. It contains a multi-level anomaly detector to monitor all three levels of network traffic data passing by. Appropriate anomaly detection methods are applied based on the distinct characteristics of data in various levels and alerts are generated, aggregated, prioritized, and sent back to control centers when anomalies are detected.

The contributions of this work are as follows:

- We divide traffic data into multiple levels and apply appropriate anomaly detection mechanism to data in each level based on their characteristics.
- We introduce the concept of confidence into the anomaly detection process and assign confidence scores to generated alerts.
- We aggregate and prioritize alerts to benefit further analysis.

The remainder of this chapter is organized as follows: Section 5.2 reviews the related work. Section 5.3 introduces the network architecture of SCADA systems and two of our design decisions. Section 5.4 gives an overview of the design of EDMAND. Section 5.6 shows the performance evaluation of EDMAND and Section 5.7 concludes the chapter.

5.2 RELATED WORK

As we mentioned in the previous section, SCADA network traffic data can be categorized into three levels but most existing network-based intrusion detection only take one or two levels into consideration. [68, 69] focus on flow-level data while [70, 71, 72, 73, 74] analyze ICS protocol functions. [75, 76] only concentrate on content-level and [77, 78] cover the flow and operation levels. None of these approaches analyze all three levels of data and therefore may fail to detect anomalies in levels not covered. Moreover, a sophisticated multi-step attack may introduce anomalies in multiple levels of traffic data. The whole picture of the attack can be seen only when all three levels of anomalies are detected.

[79] touches three levels of network traffic data by proposing an intrusion detection system implementing intelligent packet inspection mechanism, tailored traffic flow analysis, and unique packet tampering detection. However, the authors only use signature-based detection to detect malformed packets in the operation level and fail to take advantage of the periodicity of operations. In the content level, it only checks consistency of data at multiple

locations of the system but leaves the statistics of out. As a result, this approach can only deal with data tampering attacks in communication but fails to detect attacks such as fake data due to compromised devices.

The most similar work to ours is [80]. The authors develop a multiattribute SCADA-specific intrusion detection system. The system uses white lists and behavior-based rules to analyze multiple attributes in transport, operation, and content levels to mitigate various cyberattacks. However, without any method to filter and prioritize alerts, the operator can easily be overwhelmed by false positives or low-priority alerts and miss the high-priority ones. Also, in our framework, alerts in on one level will affect the alert triggering in the other two levels, which is helpful in reducing false alerts.

5.3 NETWORK ARCHITECTURE AND DESIGN DECISION

In this section, we introduce the SCADA network architecture. Then we explain two important design decisions we made for the framework.

5.3.1 Network Architecture

A simplified architecture of SCADA network is shown in Figure 5.1. The major components in SCADA network include the Master Terminal Units (MTUs) in the control centers, field controllers in the substations and the communication network that connects them. The field controllers can be Remote Terminal Units (RTUs) or Programmable Logic Controllers (PLCs), which further connect to and receive measurements from field devices such as sensors or actuators. The MTU in the control center queries the field controllers for system updates and may also issue control commands to them to change the control strategy. To avoid further data collection time and achieve prompt anomaly detection, we deploy EDMAND at the edge of the SCADA network. To be more specific, EDMAND is deployed in each substation between the field controllers and the wide area network. EDMAND monitors all traffic passing by and sends alerts back to control centers when anomalies are detected.

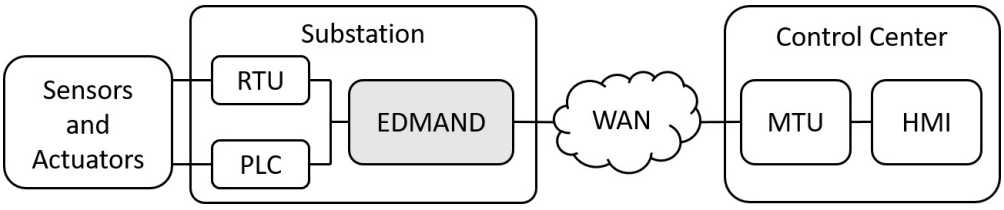


Figure 5.1: SCADA network architecture

5.3.2 Design Decision

We made two important decisions while designing our framework. The first one is to divide traffic data into multiple levels and apply appropriate anomaly detection mechanisms to data in each level based on their characteristics. As we mentioned previously, data in SCADA traffic can be divided into three levels: transport level, operation level, and content level. Data in each level have their own characteristics, which is taken into consideration when we select anomaly detection mechanisms for each level.

The second design decision is to introduce the concept of confidence into the anomaly detection process and assign confidence scores to generated alerts. We define an alert's confidence score $CS \in [0, 1]$ to be the confidence that the alert is indeed an anomaly. We calculate the confidence score by $CS = MA \times AS$, where MA is the model accuracy and AS is the anomaly score. The model accuracy measures the accuracy of our anomaly detection model in describing normal behavior and serves as the weight of the anomaly score. We assume that the majority of traffic data is normal data and therefore we can build models with higher accuracy as more samples are observed. In this sense, we estimate the model accuracy by a modified sigmoid function of observed sample number as $MA = 2/(1 + e^{-n/N}) - 1$, where n is the observed sample number by the model and $N = 100$ is a normalization factor. The anomaly score measures how far the current sample deviates from the normal behavior described by the model. Different methods are used to calculate the anomaly score for different data and they are introduced in the next section.

5.4 FRAMEWORK DESIGN

In this section, we present an overview of the modular design of EDMAND. As it is shown in Figure 5.2, EDMAND consists of 3 main components: (1)*Data Extractor*, (2)*Anomaly Detector*, (3)*Alert Manager*. The data extractor is implemented utilizing a network security monitor called Bro [39]. The data extractor monitors the network traffic passing by and forwards all three levels of network traffic data to the anomaly detector. The anomaly detector contains three levels and each level uses appropriate method to detect anomalies and generates alerts. After that, the alert manager aggregates similar alerts into meta-alerts. Priorities are given to meta-alerts and the alert manager reports meta-alerts to the control center with various frequencies according to their priorities. The anomaly detector and the alert manager will be described in more detail in the following two subsections.

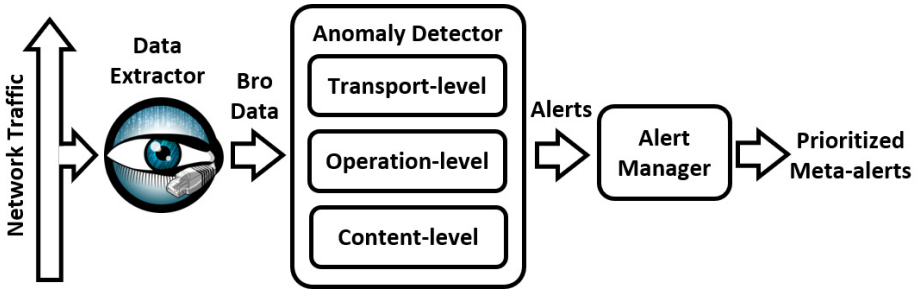


Figure 5.2: EDMAND architecture

5.4.1 Anomaly Detector

The structure of the multi-level anomaly detector is shown in Figure 5.3. There is a listener which receives Bro data from the data extractor and feeds them to the three modules for three levels. In each module, there is a parser that parses the Bro data corresponding to that level and translates them to standard input data for the processor. The processors implement various anomaly detection mechanisms to detect anomalies and generate alerts. We will introduce the three modules for three levels of data respectively.

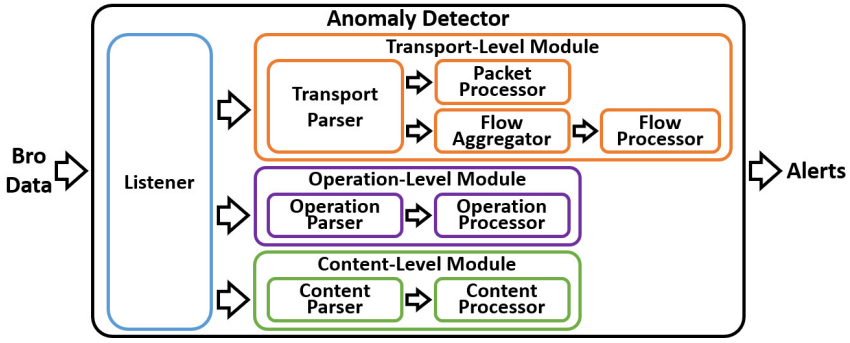


Figure 5.3: Multi-level anomaly detector structure

Transport-Level Module

In the transport-level module, two kinds of analysis at different time scales are applied. A packet processor analyzes each packet for short-term analysis. A flow aggregator aggregates packet statistics every period $T_{flow} = 10\text{min}$ and forwards to a flow processor for long-term analysis.

The input data to both processors consists of two kinds of fields: the index field which describes the packet or flow related with the input data, and data fields which store statistics for anomaly detection. As it is listed in Table 5.1, the index fields for both processors share

the same structure, which is a 4-tuple including originator(IP), responder(IP), transport protocol, and port number. The packet processor has interarrival time IAT and packet size PS as its data fields and the flow processor has packet count PC and average packet size APS as its data fields. Each type of data field has two values, corresponding to statistics of traffic in both directions.

	Packet Processor	Flow Processor
Index Field	(originator, responder, transport protocol, port number)	
Data Field	interarrival time (IAT) packet size (PS)	packet count (PC) average packet size (APS)
Mechanism	1D-DenStream	Mean-STD

Table 5.1: Input fields and anomaly detection mechanism of packet processor and flow processor

There are two types of anomalies for these two processors. The first type happens when input data with new index field are seen and the anomaly score is set to 1 in this scenario. The second type is abnormal value in data fields and we use various anomaly detection mechanisms to detect anomalies of this type. We mentioned previously that one of the design decision we made is to apply appropriate method to data with different characteristics. Since packet statistics and flow statistics follow quite different distributions, different anomaly detection mechanisms are utilized for the packet processor and flow processor. On the one hand, since traffic in SCADA usually follows periodic patterns [81, 82, 83, 84], the packet count PC and average packet size APS in a certain period usually follows a unimodal distribution as long as the period is selected properly. Therefore, the mean and standard deviation are good enough to characterize these data fields. We build models for these data fields by calculating the exponentially-weighted mean and standard deviation. The anomaly score AS is calculated as the square of the anomaly score we used in [18] as

$$AS(X, \mu, \sigma) = \begin{cases} \left(1 - \frac{\sigma^2}{|X-\mu|^2}\right)^2 & \text{if } |X - \mu| > \sigma \\ 0 & \text{otherwise} \end{cases}, \tag{5.1}$$

where μ and σ are the mean and standard deviation stored in the model and X is the data field value of the current input (i.e., PC or APS for the flow processor). For convenience, we call this anomaly detection mechanism Mean-STD in the rest of this chapter. On the other hand, the interarrival time IAT and the packet size PS of each packet usually follow multimodal distributions. Consider the following scenario, the control center is sending periodic read requests to a field controller. Each request is followed by a response from

the field controller and then a confirmation from the control center. For packets from the control center to the field controller, read requests and confirmations could have big difference in packet size PS but both are considered as normal packets. For this reason, the mean and standard deviation may not be able to characterize these data fields and we utilize a clustering method instead. We use a modified 1D version of the DenStream in [85]. DenStream is an approach to cluster data in an evolving data stream and data is clustered into potential core-micro-clusters and outlier micro-clusters. An alert is generated whenever the new value point is added to an outlier micro-cluster and the anomaly score is calculated as $AS(w, \mu, \beta) = 1 - (w - 1)/(\beta\mu - 1)$, where μ and β are predefined parameters and w is the weight of the outlier micro-cluster.

Operation-Level Module

The objective of the operation-level module of the anomaly detector is to detect anomalies in operations (e.g., requests and responses) of ICS protocols such as Modbus and DNP3. Similarly, the input data of the operation processor have an index field and a data field. We use a 5-tuple of (originator(IP), responder(IP), ICS protocol, unit id, function code) as the index field and interarrival time IAT as the data field. Here unit id is a ICS protocol specific address which is used to differential devices that share the same IP address. Notice that the IAT in operation level is different from the IAT in transport level. In operation level, the IAT is the difference in timestamps of two consecutive same operations between one pair of nodes (i.e., the two operations share the same index field). In transport level, the IAT is the difference in timestamps of two consecutive packets of the same direction between one pair of nodes which could be different operations or even non-ICS-protocol packets.

As it is shown in Table 5.2, there are mainly three types of anomalies in this level. The first type includes invalid function code and wrong direction of operation. In normal status, requests should only be sent by the control center and received by field controllers and responses should be sent by field controllers and received by the control center. Wrong direction here stands for unexpected scenarios such as requests initiated by field controllers or responses sent by the control center. For an anomaly of the first type, an alert is generated directly and a confidence scores of 1 is assigned. The second type of anomaly is the emerging of new operation, which is identified when input with new field index is observed. In this case, an anomaly score of 1 is given. The third type of anomalies includes scenarios of periodic operation arriving too early, arriving too late, or disappearing[1]. In SCADA, the IAT of

[1]The disappearing of traffic or operation is sometimes called a passive anomaly and we use a timer to detect it in both transport level and operation level.

the same operation follows a unimodal distribution since operations are usually periodic. Therefore, the Mean-STD mechanism is used for anomaly detection and AS is calculated by equation 5.1 where X is replaced by IAT in operation level.

Anomaly	Mechanism
invalid function code	CS=1
wrong direction of operation	
new operation	AS=1
early operation	Mean-STD
late operation	
missing operation	

Table 5.2: Anomaly and detection mechanism in operation level

Content-Level Module

The content-level module of the anomaly detector is responsible for detecting anomalies in measurement values such as frequencies and voltages which are included in responses to read requests. The input data of the content processor have a 5-tuple of (measure source (IP), ICS protocol, unit id, measurement type, measurement index) as the index field and the measurement value itself as the data field. Depending on the measurement type, different methods are applied for anomaly detection. Let us take DNP3 for example, where the three measurement types are Binary, Analog, and Counter. Here we will discuss the first two which are most commonly seen.

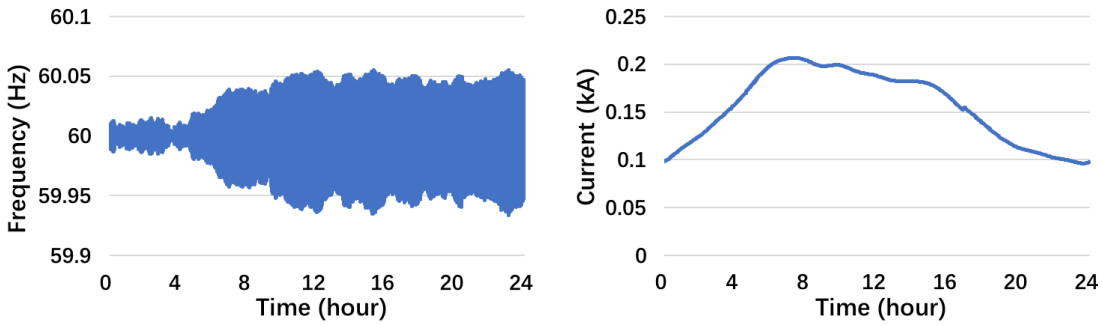
For the Binary measurement type, the intuition behind the detection method is that a binary variable can only take two values (i.e., **0** or **1**) and always one of them is normal and the other is abnormal. Therefore, we can try to identify the normal value by simply counting the **0**s and **1**s in observed samples. The normal value is **0** if the majority of the observed values are **0**s and vice versa. Whenever the abnormal value appears, we calculate the anomaly score by one minus the entropy of observed samples as

$$AS(\gamma) = \begin{cases} 1 + \gamma \log_2 \gamma + (1-\gamma)\log_2(1-\gamma) & \text{if } 0 < \gamma < 1 \\ 1 & \text{if } \gamma = 0 \text{ or } 1 \end{cases}$$

$$\text{where } \gamma = \frac{\text{number of } \mathbf{0}\text{s observed}}{\text{number of samples observed}}.$$

For the analog measurement type, we take the Smart Grid as an example. Frequency, voltage, current, and power are four most common classes of analog measurements and they

usually have quite different characteristics. The two subfigures in Figure 5.4 show one day of simulated frequency and current measurements from the Information Trust Institute's testbed [86]. We can see that 'frequency' is always around 60Hz and has a very small relative standard deviation whereas 'current' varies a lot but follows a diurnal pattern. Based on different analog classes' characteristics, we develop a 2-step anomaly detection method to analog measurements. In step 1, we further categorize analog measurements into different analog classes (i.e., frequency, voltage, current, power) and use an appropriate method for each class to detect anomalies in step 2. Notice that if the configuration files of field controllers are available and the specification of each analog index is known to our framework, step 1 can be neglected. Step 1 just provides analog class inference ability to our framework when specification is not given.



(a) Frequency (b) Current

Figure 5.4: One day's simulated measurement data of frequency and current

More specifically, in step 1 of our analog measurement anomaly detection, we utilize a similar Bayesian inference method as in [87] and build an analog class inference model based on the Bayesian network. Here we use a very simple Bayesian network with one root node and three leaf nodes shown in Figure 5.5. Each leaf node has a conditional probability table (CPT) representing the prior knowledge of the dependence between the child node and its parent node whose elements are defined by $CPT_{ij} = P(child\_state = j | parent\_state = i)$. The root represents the analog class with four hypothesis states and the leaf nodes represent directly observable evidences and each leaf node has several discrete states. The objective of this model is to calculate the belief in hypotheses of the root, which is decided by the likelihood propagated from its child nodes and ultimately the observed evidences at the leaf nodes. We denote y^k $(k \in 1 \dots 3)$ as the observation at k^{th} leaf node and x_i $(i \in 1 \dots 4)$ as the i^{th} analog class at the root node. Let $P(x_i)$ be the prior probability for the hypotheses of the root. The prior probability and CPTs can either be acquired based on domain knowledge or calculated based on training data. The believe in the analog class x_i is represented by the

conditional probability of x_i given the observation at all leaf nodes and can be calculated by

$$P\left(x_i|y^1, y^2, y^3\right) = \alpha P(x_i) \prod_{k=1}^{3} P\left(y^k|x_i\right),$$

where $\alpha = 1/P\left(y^1, y^2, y^3\right)$ and can be calculated by $\sum_i P\left(x_i|y^1, y^2, y^3\right) = 1$. If the believe of x_i is larger than a threshold $\theta_b = 0.7$, the inference model infers the analog measurement as class x_i. Since the analog class for the same series of measurements will not change in normal status, the inference model stops analyzing for that series of measurements after its class is successfully inferred.

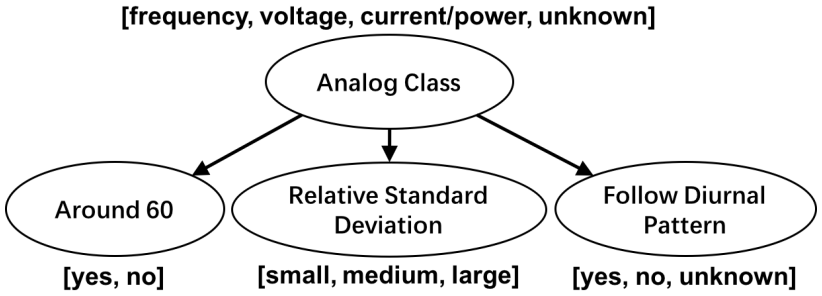


Figure 5.5: Analog class inference model

After the analog classes are inferred, different anomaly detection methods are applied as shown in Table 5.3. Mean and standard deviation are used for frequency and voltage. For current or power, we divide 24 hours to multiple time slots and calculate mean and standard deviation in each of the slot throughout multiple days. For analog measurements not belonging to any of the mentioned classes, the 1D-DenStream method for the flow processor is utilized. Notice that this list of analog classes can be extended by incorporating prior knowledge of other analog classes into the inference model and taking their characteristics into consideration while selecting their anomaly detection methods.

Analog Class	Mechanism
frequency	Mean-STD
voltage	
current/power	slotted Mean-STD
unknown	1D-DenStream

Table 5.3: Analog measurement class and detection mechanism

5.4.2 Alert Manager

The alerts generated by EDMAND's multi-level anomaly detector which have confidence score higher than a threshold θ_{CS} are forwarded into the alert manager. We use a dynamic mechanism for θ_{CS}. The initial threshold is set at a upper bound value of $\theta_{CS\_H} = 0.95$. For every alert with CS above the current threshold, we calculate the new threshold as $\theta_{CS\_new} = e^{-\lambda}(\theta_{CS\_cur} - \theta_{CS\_L}) + \theta_{CS\_L}$ where $\theta_{CS\_L} = 0.85$ is a lower bound and $\lambda = 0.05$ is a parameter can be tuned. While we exponentially decrease the threshold to approach the lower bound for triggered alerts, we also linearly increases the threshold to the upper bound as time goes by. Keeping an high threshold in normal state helps to decrease false alarms and decreasing the threshold when alarms are triggered helps to detect all anomalies related to the attack and is useful against multi-step attacks. Since the alert from the three levels share the same threshold, alerts in one level will lower the threshold, making it easier to detect simultaneous anomalies in the other two levels if there are any.

Alerts generated by different processors share the following common fields: index field, alert type, timestamp, confidence score, statistical fields and abnormal data. Index field is the same as the index field in the input data of the corresponding processor. Alert type is the description of the anomaly. Statistical fields include statistics in the data field such as current data value, mean, standard deviation, etc. Abnormal data is the original input data of the processor which triggering the alert. As it is shown in Figure 5.6, the alert manager consists of two components: the *Alert Aggregator* and the *Alert Scheduler*.

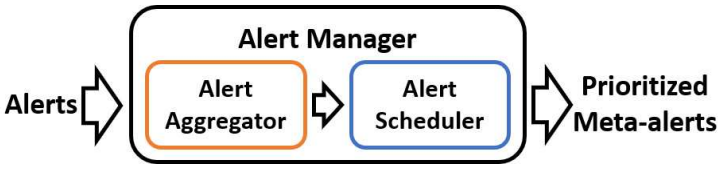


Figure 5.6: Alert manager structure

The objective of the alert aggregator is to aggregate alerts that share the same alert type as well as the index field and have little difference in timestamps. The aggregated alert is called the meta-alert. The meta-alert inherits all the fields from the alerts before aggregation with each field type having its own aggregation rule listed in Table 5.4. Another count field is added to store the number of alerts aggregated to this meta-alert. Whenever the alert aggregator receives a new alert, it tries to aggregate it to existing meta-alerts. If there is no meta-alert that this alert can merge to, a new meta-alert is created. In this way, consecutive duplicate alerts about the same event are aggregated to one meta-alert, which prevents alert flooding and simplifies further analysis of the alerts.

Alert Field	Aggregation Rule
index field	shared by all of the aggregated alerts
alert type	
timestamp	keep minimum and maximum
confidence score	keep maximum
statistical fields	inherit from the last alert aggregated
anomaly data	
count	number of aggregated alerts

Table 5.4: Meta-alert fields and aggregation rules

Every time a meta-alert is created or updated, it is forwarded to the alert scheduler, where its priority score is calculated and its report frequency is decided. The alert priority computation model in Figure 5.7 is similar to the analog class inference model. It is a Bayesian network with one root node and five leaf nodes. The root represents the alert priority which has two hypothesis states of low and high. Similarly, the leaf nodes represent observable evidence and each has several discrete states. We denote y^k $(k \in 1 \dots 5)$ as the observation at k^{th} leaf node. We define the priority score PS as $PS = P\left(Priority = high | y^1, y^2, y^3, y^4, y^5\right)$ which can be calculated in a similar way as the analog class inference model. The prior probability of priority, CPTs at leaf nodes, and criteria to categorize observation are set by domain knowledge or system requirements. For example, the critical operation set for DNP3 can be defined by the DNP3 critical request function codes in [88] or entered by utilities according to their own needs.

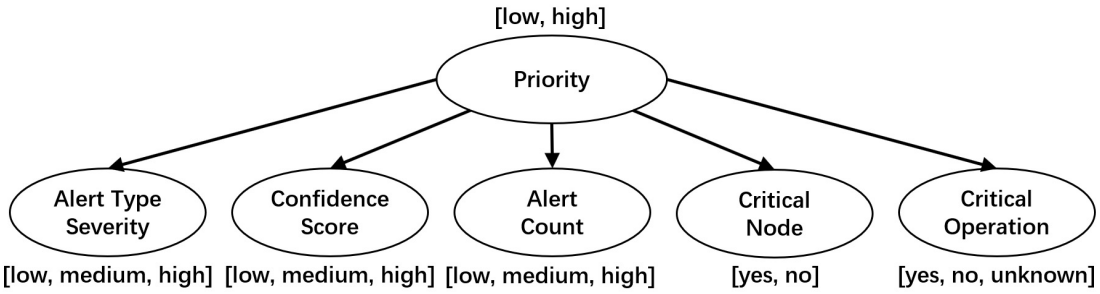


Figure 5.7: Alert priority computation model

Different report mechanisms in Table 5.5 are applied to meta-alerts based on their priority scores PS. After PS is calculated, the meta-alert is classified as high-priority or low-priority, based on PS and a threshold $\theta_p = 0.7$ as shown in Table 5.5. High-priority meta-alerts are always reported immediately when first created and reported with a small period if they are updated during that period. Low-priority meta-alerts are not reported immediately upon creation and reported with a large period if they are updated during the period. In the

future, we plan to design a causality-based anomaly analyzer in the control center to further correlate and analyze the received alerts.

	High-Priority	Low-Priority
Definition	$PS \geq \theta_p$	$PS < \theta_p$
Report when first created	yes	no
Report period	T_h	$T_l (> T_h)$

Table 5.5: Meta-alert report mechanism

5.5 DISCUSSION

In this section, we want to briefly discuss the scalability and extensibility of EDMAND. There are multiple substations in SCADA networks and each substation contains an EDMAND by default. So adding more substations will not cause any scalability issue of EDMAND. Increasing the number of end devices in each substation does increase the complexity of the network traffic, and further increases the anomaly detection complexity of EDMAND. However, the number of end devices in each substations is usually quite limited. For example, Modbus's unit id has a range of 0-247, which limits the number of end devices of each substation. Even if the number of end devices at each substation is too many for EDMAND to handle in real time, there is nothing to prevent us from deploying multiple EDMANDs in each substation. Moreover, the packet analysis of EDMAND happens in parallel with the packet passing and does not block the packets. Therefore, the original communication in SCADA will not be affected by EDMAND.

The current prototype of EDMAND supports Modbus and DNP3 for SCADA networks. To add support for other industrial control protocols, only the data extractor needs to be extended. As long as the data extractor is equipped with the capabilities of parsing desired protocols and the outputs follow a consistent format, the other components need not to be changed to support those protocols.

5.6 PERFORMANCE EVALUATION

In this section, we evaluate the performance of EDMAND in two aspects: detection ability and time overhead. The data extractor is implemented by Bro and the anomaly detector as well as the alert manager are implemented in Python. The evaluation is based on a simulated DNP3 traffic set which includes periodic baseline traffic and injected anomalies in

transport, operation, and content levels. The baseline traffic consists of 10 days of simulated traffic of one control center sending read requests to two field controllers every 20 seconds. Each read request is followed by a TCP acknowledgement as well as a response from the field controller. After the response, which contains the requested measurements, is received by the control center, the control center sends a confirmation which is again followed by an acknowledgement from the field controller. The baseline traffic is generated based on DNP3 specification [25], domain experts' advices, and some example DNP3 traffic we have generated by a testbed from Information Trust Institute at Illinois. Each field controller contains 5 measurements: one binary measurement and four analog measurements including frequency, voltage, current and power. Those measurements are also simulated data from the testbed.

The analog class inference model correctly identifies all analog classes in the baseline traffic. We inject various anomalies in the three levels listed in Table 5.6 to evaluate EDMAND's anomaly detection ability. EDMAND is able to detect all the anomalies injected with no false alarms. This 0% false alarm rate has two reasons. The first one is that EDMAND already filters out low-quality alerts by discarding alerts with low confidence scores. And the second reason is that the simulated traffic is not as noisy as real-world SCADA traffic. We consider to use real traffic with real attacks to evaluate EDMAND if we can get access to real data in the future. All the injected anomalies generate 12135 alerts in total, and are aggregated to 22 meta-alerts. We list in Table 5.6 some related works' detection abilities based on their description for anomalies which we injected. None of them are able to detect all anomalies like EDMAND does.

We also create a simple multi-step attack scenario with simulated traffic. In step 1, the attacker scans several ports in a given IP address range to find the target field device and the ICS protocol which the SCADA system is using. In step 2, the attacker sends a write request to the field device to compromise the device. In step 3, the compromised device sends tampered data in responses to read requests from the control center. EDMAND is able to detect all three steps of the attack where the three steps are detected in transport level, operation level, and content levels, respectively. A framework, concentrating on one or two levels of data analysis only, may not be able to see the whole picture of the attack.

One of our design decisions is to apply appropriate anomaly detection mechanisms to data, based on their characteristics. We use the 1D-DenStream mechanism for the packet processor since its data fields (i.e., interarrival time and packet size) follow multimodal distributions. To validate this design decision, we use the Mean-STD mechanism instead for the packet processor. We find that the modified packet processor is unable to detect the padded response and the delayed TCP acknowledgement, and generates tons of false

Level	Anomaly	Detection Ability					
		ED	[68]	[73]	[75]	[77]	[80]
Transport	add a new node to send several packets to one field controller	Y	Y	N	N	Y	Y
	pad one response from a field controller with more payload	Y	Y	N	N	Y	Y
	delay one TCP acknowledgement from a field controller intentionally	Y	Y	N	N	N	N
	send lots of ICMP packets in a short period to one field controller	Y	Y	N	N	Y	Y
Operation	send one operation with invalid function code to one field controller	Y	N	Y	M	Y	Y
	let one field controller send a control command to the control center	Y	N	Y	M	Y	Y
	delay one response from a field controller intentionally	Y	M	N	N	N	Y
Content	tamper with the binary value from one field controller for a short period	Y	N	N	Y	N	Y
	introduce over voltage and under voltage tripping to voltage measurements	Y	N	N	Y	N	Y
	introduce over current tripping to current measurements	Y	N	N	M	N	M
	tamper with the frequency value from one field controller for a short period	Y	N	N	Y	N	Y
	tamper with the power value from one field controller for a short period	Y	N	N	M	N	M

Table 5.6: Injected anomalies and detection ability comparison (ED=EDMAND, Y=yes, N=no, M=maybe)

alarms. This proves that selecting appropriate anomaly detection mechanism according to data characteristics is important. We also validate the alert priority computation model by calculating priority scores of meta-alerts triggered by two anomalies. The first anomaly is that the control center suddenly starts to send periodic write request (critical operation) to the field controller, which is considered a critical node. The second anomaly is the delay of one TCP acknowledgement from a field controller which is not a critical node. The meta-alert for the first anomaly has a priority score of 0.995, which is higher than the score of 0.439 for the second anomaly. This is consistent with the fact that the first anomaly is more critical than the second one.

To demonstrate EDMAND's ability to satisfy the real-time requirements of anomaly detection in SCADA systems, we also evaluate the time overhead of data analysis in the three

levels and the alert manager. We run our experiments on the Ubuntu 16.04 desktop with 12 Intel Xeon 3.60GHz CPUs and 16GB memory. The data extraction in Bro and the anomaly detection for the three levels run in parallel. The total analysis time (data extraction time + anomaly detection time) per packet is 3.87ms for transport level, 6.66ms for operation level, and 1.94ms for content level. These time overheads are comparable with previous works [33, 69] which also have a time overhead in the order of a few milliseconds per packet. Also, since the common data collection interval in SCADA systems is seconds or even minutes [89], several milliseconds overhead per packet is short enough for the packets to be processed at communication line speed. The average time overhead of the anomaly manager for each alert is 423ms. The rate of alerts varies a lot for different attacks. Usually, the rate of alerts is far smaller than the rate of packets. For example, in [69], the man-in-the-middle attack generates 4195 alerts per hour per RTU. EDMAND is able to handle all the alerts in real time in this case. If there is a burst of alerts for some specific attacks, then most of the alerts could be duplicates that contain redundant information. Therefore, dropping some of them will not affect the further analysis of the alerts and can help EDMAND to process the rest of the alerts in real time.

5.7 CONCLUSION

In this chapter, I present EDMAND, an edge-based multi-level anomaly detection framework for SCADA systems. EDMAND resides in remote substations of SCADA systems and monitors network traffic of flow level, operation level, and content level. Distinct data characteristics are considered when selecting anomaly detection method for each level. When anomalies are detected, EDMAND generates, aggregates, and prioritizes alerts and sends them to control centers. The performance of EDMAND is validated by experiments.

CHAPTER 6: CAPTAR: CAUSAL-POLYTREE-BASED ANOMALY REASONING FOR SCADA NETWORKS

6.1 INTRODUCTION

Nowadays, large-scale distributed critical infrastructure systems such as power grids and refineries increasingly rely on digital industrial control systems (ICSs) for real-time monitoring, data collection, and control. The Supervisory Control and Data Acquisition (SCADA) system is the most commonly used ICS. Critical as they are, SCADA systems are subject to a wide range of serious threats for reasons mentioned in Section 5.1. Therefore, securing SCADA systems against various threats and vulnerabilities has become a major challenge.

To promote the security of SCADA systems, intrusion detection systems (IDSs) are increasingly deployed by SCADA operators. As the name suggests, the main objective of IDSs is to monitor the system, detect suspicious activities caused by intrusion attempts, and report alerts to the system operators. Although IDSs play an undeniable role in the protection of SCADA systems, they still suffer from some defects. The biggest issue with traditional IDSs is that they continuously generate tremendous number of alerts without further comprehending them. Drowned in an ocean of unstructured alerts mixed with false positives, SCADA operators are almost blind to see any useful information. Due to the high volume and low quality of the alerts, it becomes a nearly impossible task for the operators to figure out the complete pictures of the attacks and take appropriate actions in a timely manner.

To address the aforementioned problem of traditional IDSs and provide the SCADA operators with *explainable situational awareness*, there is a need for an efficient system to aggregate redundant alerts from IDSs, correlate them in an intelligent manner, and discover attack strategies based on domain knowledge as well as causal reasoning. In Chapter 5, we described our edge-based multi-level anomaly detection framework for SCADA, named EDMAND. EDMAND resides at the edges of the SCADA network and detects anomalies in multiple levels of the network. The triggered alerts are aggregated, prioritized, and sent to the control center. In this chapter, we present a causal-polytree-based anomaly reasoning framework for SCADA networks, named CAPTAR. CAPTAR resides in the control center of the SCADA network and takes the meta-alerts from EDMAND as input (shown in Figure 6.1). CAPTAR correlates the alerts using a naive Bayes classifier and matches them to predefined causal polytrees. Utilizing Bayesian inference on the causal polytrees, CAPTAR is able to reveal the attack scenarios from the alerts and produces a high-level view of the security state of the protected SCADA network.

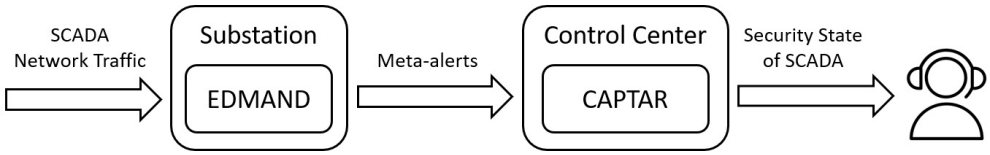


Figure 6.1: Locations of EDMAND and CAPTAR

The remainder of this chapter is organized as follows: Section 6.2 reviews the related work. Section 6.3 introduces the basic concept of Bayesian network, Bayesian inference, and belief propagation. These concepts are utilized in the anomaly reasoning in this chapter. Two canonical models which are used to build our causal polytrees are also introduced in Section 6.3. Section 6.4 gives an overview of the design of CAPTAR. Section 6.5 shows the performance evaluation of CAPTAR and Section 6.6 concludes the chapter. Since this Chapter uses many difference notations, some of the most important notations are listed in Table 6.1 for quick reference.

6.2 RELATED WORK

Various techniques have been used to measure the similarity of common features of alerts to correlate them [90, 91, 92, 93]. However, alert correlation alone can only measure the correlation strength between alerts and are not sufficient to recognize the whole picture of the attack.

To fill the gap of alert correlation, many works have been proposed in the area of attack plan recognition. Some works [94, 95] keep the state of the system and assume that the state evolves towards a "worse" direction during attacks. There are also works [96, 97] that define prerequisites and consequences of each attack step and construct chains or graphs based on the matching of prerequisites and consequences. Bayesian networks are also utilized by many papers [98, 96, 99, 100, 101] to correlate alerts or to represent and infer the causal relationship between attack steps.

The closest previous work [87] to ours is the integration of alert aggregation, prioritization, correlation, and attack plan recognition. Three alert correlation methods are proposed: probabilistic-based, causal discovery-based, and temporal based methods. The attack plan recognition step also uses causal polytrees to represent attack plans.

CAPTAR mainly differentiates from all previous works in two aspects. First, the alerts received by CAPTAR are meta-alerts generated by EDMAND, which is our edge-based multi-level anomaly detection framework for SCADA. EDMAND applies network-based rather than host-based detection and it mainly takes the anomaly-based approach instead of the

Notation	Description
X	A node in the Bayesian network representing a random variable.
U	A parent of X in the Bayesian network.
Y	A child of X in the Bayesian network.
$\boldsymbol{e}_X^+$	Evidence contained in the sub-tree rooted at X.
$\boldsymbol{e}_X^-$	Evidence contained in the rest of the Bayesian network other than $\boldsymbol{e}_X^+$.
$\pi_X(u)$	Causal support provided by parent U to X.
$\lambda_Y(x)$	Diagnostic support provided by child Y to X.
$BEL(x)$	Belief at node X.
$\widetilde{X}$	Auxiliary child node of X to simulate evidence of matched meta-alerts.
$\lambda_{\widetilde{X}}(x)$	Diagnostic support provided by $\widetilde{X}$ to X.
I	Inhibitory mechanism in "noisy-OR" model.
E	Enabling mechanism in "noisy-AND" model.
q	Probability that the inhibitory or enabling mechanism is active.
$CS(a)$	Confidence score of meta-alert a.
AT	Attack template.
$\boldsymbol{ATS}$	Attack template set.
$\boldsymbol{S}_{AT}$	Set of consequence nodes of attack template AT.
AU	Alert unit.
w	Weight in the alert unit.
A	Alert type in the alert unit.
CS_{total}	Total confidence score of all matched meta-alerts of a node.
$BEL_{max}(AT)$	Maximum probability of existence of all consequence nodes in AT.
Cor_{max}	Maximum correlation score of a meta-alert in an attack template.
$\boldsymbol{ATS}_{match}$	Attack template set containing alert matching results.
X_{cor}	Exact match node with the highest correlation score for a meta-alert.
$\boldsymbol{X}_{pot}$	Set of potential nodes a meta-alert could match to.
K	Maximum number of attack templates to keep for each kind of attack.
M	Number of meta-alerts in the meta-alert database.
N	Maximum number of nodes in any attack template.
L	Number of attack templates in the attack template database.

Table 6.1: Table of notation for Chapter 6

signature-based approach. The alerts from EDMAND do not directly relate to each attack step in the attack plan but instead relate to various network behaviors triggered by each attack step. Therefore, mapping between alerts from EDMAND and underlying attack steps is necessary for our anomaly reasoning. Second, we define the concept of confidence score for each alert in EDMAND. In CAPTAR, the confidence scores of meta-alerts are utilized to calculate the diagnostic support for each node in the causal polytrees during the belief propagation. This allows each alert to carry more belief information instead of only a binary state (exist/not exist).

6.3 PRELIMINARIES

An example workflow of EDMAND and CAPTAR is shown in Figure 6.2. After an attacker launches an attack, each step of the attack could result in one or more anomalies in the network traffic. These anomalies trigger meta-alerts in EDMAND. CAPTAR receives the meta-alerts from EDMAND and tries to infer the attack steps that triggered them by mapping meta-alerts to attack steps. Potential attack steps are structured as nodes in causal polytrees whose nature are Bayesian networks. Bayesian inference is performed on those causal polytrees to reason about the existence of attacks. In this section, we first introduce the basic concept of Bayesian network. Then we describe inference in Bayesian network followed by the belief propagation algorithm to conduct Bayesian inference. Finally, we present two canonical models we use to build our causal trees: "noisy-OR" and"noisy-AND" models.

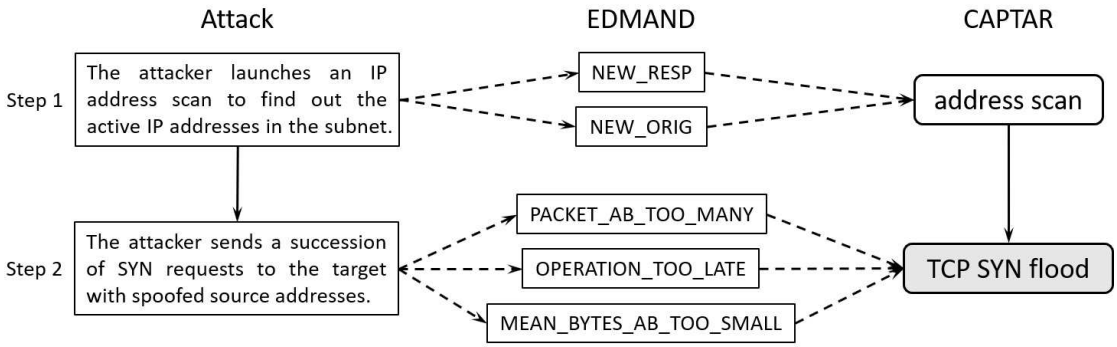


Figure 6.2: An example workflow of EDMAND and CAPTAR

6.3.1 Bayesian Network

Before going into exactly what a Bayesian network is, it is first useful to review two concepts in probability theory. The first concept is the chain rule of probability. It says that if we have a set of n random variables, $X_1, X_2, \ldots, X_n$, then the joint probability distribution $P(X_1, X_2, \ldots, X_n)$ can be written as a product of n conditional probabilities:

$$P(X_1, X_2, \ldots, X_n) = P(X_n|X_{n-1}, \ldots, X_2, X_1) \cdots P(X_2|X_1)P(X_1). \tag{6.1}$$

The second concept is the conditional independence. We say that two random variables, A and B, are conditionally independent given another random variable C if $P(A|B, C) = P(A|C)$. In other words, once we know C, learning B would not change our belief in A.

After recalling the chain rule of probability and the conditional independence, we can introduce the basics of Bayesian network. A Bayesian network is a directed acyclic graph (DAG) in which the nodes represent variables, the edges signify the existence of direct causal influences between the linked variables, and the strengths of these influences are expressed by conditional probabilities. Figure 6.3 illustrates a simple yet typical Bayesian network. It describes relationships among the seasons of the year (X_1), whether rain falls (X_2), whether the sprinkler is on (X_3), whether the pavement would get wet (X_4), and whether the pavement would be slippery (X_5). All variables in this figure are binary (taking a value of either true or false) except for the root variable X_1, which can take one of four values: spring, summer, fall, or winter.

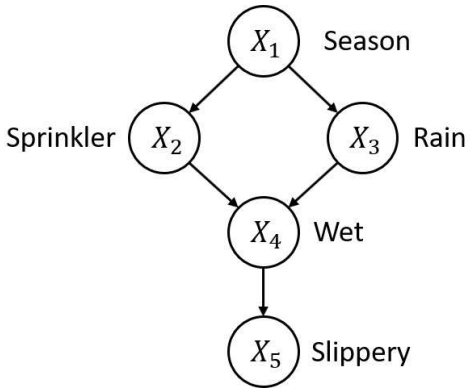


Figure 6.3: Bayesian network example

Each edge in the figure represents a direct causal influence from the head of the edge to the tail. In Figure 6.3, X_4 has a directed edge pointing to X_5. This is because the fact that the pavement is wet has a direct causal influence on whether the pavement is slippery. On the contrary, the absence of a direct edge between two nodes implies conditional independence. For example, the absence of a direct edge between X_1 and X_5 captures the understanding that the influence of seasonal variations on the slipperiness of the pavement is mediated by other conditions (e.g., the wetness of the pavement).

Each node in the Bayesian network is associated with a probability function that takes (as input) a particular set of values for the node's parent variables, and gives (as output) the probability of the variable represented by the node. The most common form of this probability function is a conditional probability table (CPT). CPT is a table defined for a set of discrete and mutually dependent random variables to display conditional probabilities of a single variable with respect to the others. An example CPT of X_4 in Figure 6.3 is shown in Table 6.2. It gives the conditional probabilities of $P(X_4|X_2, X_3)$.

An important property of Bayesian networks is the local (parental) Markov condition,

X_2	X_3	$X_4 = \mathrm{T}$	$X_4 = \mathrm{F}$
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Table 6.2: Conditional probability table of X_4

which states that every node in a Bayesian network is conditionally independent of all its non-descendants given its parent. In the above example, we have $P(X_5|X_1, X_2, X_3, X_4) = P(X_5|X_4)$ since Slippery is conditionally independent of its non-descendants, Season, Sprinkler, and Rain, given its parent Wet. This property allows us to simplify the joint distribution, obtained using the chain rule, to a simpler form. Assume a Bayesian network has n nodes $X_1, \ldots, X_n$ in total. The joint distribution can be simplified as

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|X_1, \ldots, X_{i-1}) = \prod_{i=1}^{n} P(X_i|Parents(X_i)), \tag{6.2}$$

where $Parents(X_i)$ is the set of direct parents of X_i. In the above example, we are able to rewrite the joint distribution as

$$P(X_1, X_2, X_3, X_4, X_5) = P(X_1)P(X_2|X_1)P(X_3|X_1)P(X_4|X_2, X_3)P(X_5|X_4). \tag{6.3}$$

This property significantly reduces the amount of required computation in large Bayesian networks since each node usually has fewer parents compared with the overall size of the network.

6.3.2 Bayesian Inference

There are two kinds of inference over a Bayesian network. The first is to evaluate the joint probability of a specific assignment of values for all or a subset of variables in the network. For all variables, we simply factorize the joint probability using Equation 6.2 and calculate the product using provided conditional probabilities. For a subset of variables, we marginalize over the variables not in the subset by summing up probabilities over them and get the marginal probability of the subset of variables we are interested in.

The second and more interesting inference is to evaluate $P(\boldsymbol{x}|\boldsymbol{e})$, that is, the probability of some particular assignment of a subset of variables ($\boldsymbol{x}$) given assignments of other variables (evidence $\boldsymbol{e}$). In the scenario we mentioned in Section 6.3.1, one example of this kind of

inference could be to evaluate $P(X_2 = T, X_4 = T, X_5 = T|X_1 = spring)$. In this case, $\{X_2 = T, X_4 = T, X_5 = T\}$ is our $\boldsymbol{x}$ and $\{X_1 = spring\}$ is our $\boldsymbol{e}$. According to the definition of conditional probability, we have $P(\boldsymbol{x}|\boldsymbol{e}) = P(\boldsymbol{x}, \boldsymbol{e})/P(\boldsymbol{e}) = \alpha P(\boldsymbol{x}, \boldsymbol{e})$, where $\alpha = 1/P(\boldsymbol{e})$ is a normalizing constant rendering $\sum_{\boldsymbol{x}} P(\boldsymbol{x}|\boldsymbol{e}) = 1$. Let $\boldsymbol{Z}$ represent the set of variables in the network that is not in $\boldsymbol{x}$ and $\boldsymbol{e}$, and $\boldsymbol{z}$ represents any particular value assignment of $\boldsymbol{Z}$. To get $P(\boldsymbol{x}, \boldsymbol{e})$, the marginal probability of $\{\boldsymbol{x}, \boldsymbol{e}\}$ over $\boldsymbol{Z}$ needs to be calculated. Therefore, we have

$$P(\boldsymbol{x}|\boldsymbol{e}) = \alpha \sum_{\forall \boldsymbol{z} \in \boldsymbol{Z}} P(\boldsymbol{x}, \boldsymbol{e}, \boldsymbol{z}). \tag{6.4}$$

In the example, we can calculate $P(X_2 = T, X_4 = T, X_5 = T|X_1 = spring)$ as

$$\begin{aligned}
&P(X_2 = T, X_4 = T, X_5 = T|X_1 = spring) \\
&= \alpha \sum_{X_3} P(X_1 = spring)P(X_2 = T|X_1 = spring)P(X_3|X_1 = spring) \\
&\quad P(X_4 = T|X_2 = T, X_3)P(X_5 = T|X_4 = T) \\
&= \alpha P(X_1 = spring)P(X_2 = T|X_1 = spring)P(X_3 = T|X_1 = spring) \\
&\quad P(X_4 = T|X_2 = T, X_3 = T)P(X_5 = T|X_4 = T)+ \\
&\quad \alpha P(X_1 = spring)P(X_2 = T|X_1 = spring)P(X_3 = F|X_1 = spring) \\
&\quad P(X_4 = T|X_2 = T, X_3 = F)P(X_5 = T|X_4 = T)
\end{aligned} \tag{6.5}$$

6.3.3 Belief Propagation

Belief propagation via message passing [102] is an algorithm to conduct inference on Bayesian networks. To make it clearer, we first illustrate the belief propagation rules in a general tree-structured Bayesian network where a node might have several children and one parent. In the next subsection, we will introduce the two canonical models which generalize our causal trees to polytrees.

We illustrate the belief propagation by specifying the activities of a typical node X having m children, Y_1, Y_2, ..., Y_m, and a parent U as shown in Figure 6.4. The belief in the various values of X depends on two distinct sets of evidence: evidence from the sub-tree rooted at X, and the evidence from the rest of the tree. In general, let us define $\boldsymbol{e}_X^-$ as the evidence contained in the tree rooted at X and define $\boldsymbol{e}_X^+$ as the evidence contained in the rest of the network. $\boldsymbol{e}_{Y_j}^-$ therefore represents the evidence from the sub-tree rooted at Y_j where $j \in \{1, \ldots, m\}$. $x \in \{0, 1\}$ is a particular value of X and $u \in \{0, 1\}$ is a particular value of U. The belief distribution of variable X can be calculated based on the following three

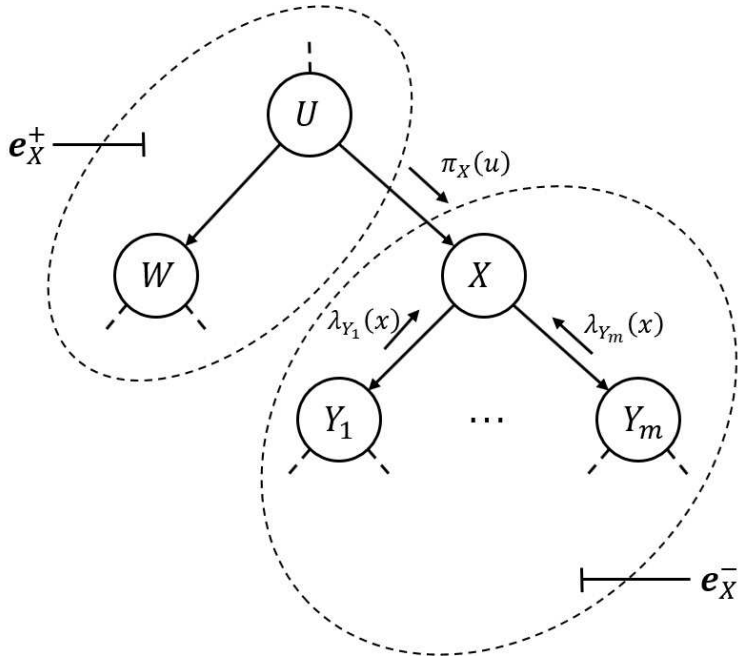


Figure 6.4: Fragment of a causal tree, showing different kinds of evidence and support of a node X

kinds of parameters:

1. *Causal* Support: $\pi_X(u) = P(u|\boldsymbol{e}_X^+)$, contributed by parent of X.

2. *Diagnostic* Support: $\lambda_{Y_j}(x) = P(\boldsymbol{e}_{Y_j}^-|x)$, contributed by the Y_j which is the j-th child of X where $j \in \{1, \ldots, m\}$.

3. Conditional Probability Table (CPT): $P(x|u)$ that relates the variable X to its direct parent U. Each entry $P(x|u)$ in the table defines the probability of value x of node X given certain value u of node U.

We utilize the tree-structured Bayesian network for our anomaly reasoning. Each node represents an attack step in the entire attack plan and it has two states of exist (1) and not exist (0). A direct edge from node U to node X means that attack step U is a direct prior step of attack step X and needs to be launched before X. In this way, we are able to reason about the probability of existence of the attack by calculating the belief of each attack step.

The belief propagation algorithm runs whenever new evidence is found in the tree. The propagation starts from the node which receives the new evidence and the new belief propagates along the edges of the tree until all nodes get updated. The local belief updating at each node X can be executed by three steps in any order.

Belief Propagation Algorithm

Step 1 — Belief updating: Node X updates its belief measure based on the $\pi_X(u)$ message from its parent and the messages $\lambda_{Y_1}(x)$, $\lambda_{Y_2}(x)$, ..., $\lambda_{Y_m}(x)$ from each of its children as shown in Figure 6.4.

$$BEL(x) = \alpha\lambda(x)\pi(x), \tag{6.6}$$

where

$$\lambda(x) = \prod_j \lambda_{Y_j}(x), \tag{6.7}$$

$$\pi(x) = \sum_u P(x|u)\pi_X(u), \tag{6.8}$$

and α is a normalizing constant rendering $\sum_x BEL(x) = 1$.

Step 2 — Bottom-up propagation: As shown in Figure 6.6, node X computes a new message $\lambda_X(u)$ based on its CPT and λ messages received from its children. Then X sends $\lambda_X(u)$ to its parent U.

$$\lambda_X(u) = \sum_x \lambda(x)P(x|u), \tag{6.9}$$

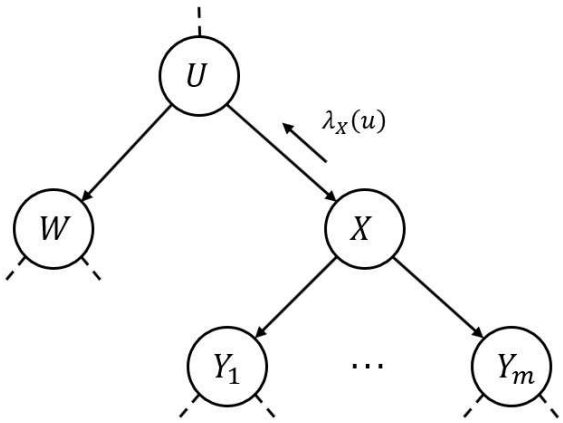


Figure 6.5: Bottom-up propagation

Step 3 — Top-down propagation: As shown in Figure 6.5, node X computes new π messages and sends them to its children. The new $\pi_{Y_j}(x)$ message for its j-th child Y_j is calculated as

$$\pi_{Y_j}(x) = \alpha\pi(x)\prod_{k\neq j} \lambda_{Y_k}(x). \tag{6.10}$$

Boundary conditions are established as follows:

1. *Root nodes:* If X is a node with no parents, we set $\pi(x)$ equal to the prior probability $P(x)$.

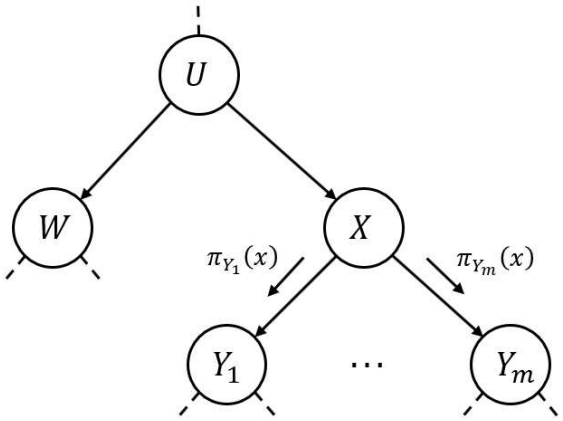


Figure 6.6: Top-down propagation

2. *Anticipatory nodes:* If X is a childless node that has not been instantiated, we set $\lambda(x) = 1$ for $x \in \{0, 1\}$

3. *Evidence nodes:* In our anomaly reasoning, evidence for a node X is obtained when meta-alerts are matched to the node. We will discuss the matching mechanism in Section 6.4.4. When evidence is obtained for X, we add a dummy auxiliary child node $\widetilde{X}$ to X as shown in Figure 6.7 and simulate the evidence by letting $\widetilde{X}$ provide a diagnostic support message $\lambda_{\widetilde{X}}(x)$ to X. We will describe our way to calculate $\lambda_{\widetilde{X}}(x)$ in Section 6.4.2. This auxiliary node $\widetilde{X}$ is not updated during the belief propagation using the 3 steps mentioned. It only changes the way X calculates its own $\lambda(x)$. Therefore, if evidence of X is obtained, Equation 6.7 needs to be rewritten as

$$\lambda(x) = \lambda_{\widetilde{X}}(x) \prod_j \lambda_{Y_j}(x). \tag{6.11}$$

6.3.4 The "noisy-OR" and"noisy-AND" Models

In Section 6.3.3, we illustrate the belief propagation algorithm in a general tree-structured Bayesian network where a node has at most one parent. However, this structure lacks the ability to represent nodes that might have multiple causes (i.e., node may have multiple parents). In this subsection, we introduce two canonical models which allow us to generalize our causal trees to causal polytrees. A polytree is a directed acyclic graph whose underlying undirected graph is a tree. An example polytree is shown in Figure 6.8. The difference between a polytree and a normal tree is that a node could have multiple parents in a polytree. The two canonical models contain structures similar to logical OR-gate and AND-gate with

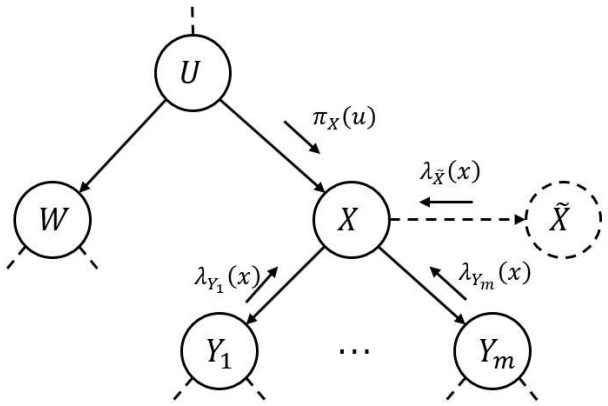


Figure 6.7: Auxiliary child $\widetilde{X}$ of X representing evidence received by X

noises and are thus called "noisy-OR" and "noisy-AND" models. The characteristics of these two typical structures enable us to conduct the belief updating more efficiently in polytrees.

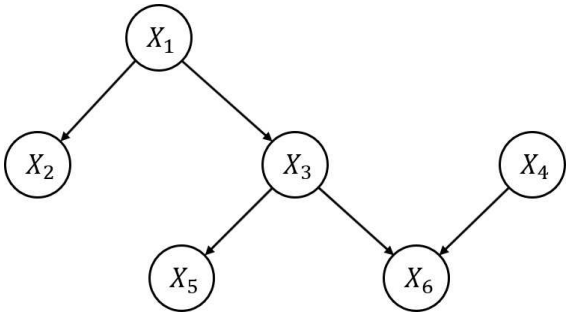


Figure 6.8: Polytree example

The "noisy-OR" Model

The "noisy-OR" model [102] is based on the noisy OR-gate structure shown in Figure 6.9. Each node represents an event (attack step in our anomaly reasoning) with binary state 0 or 1. For a node X with n parents $\boldsymbol{U} = \{U_1, U_2, \ldots, U_n\}$, its value can be seen as the output of a logical OR-gate. Each input to the OR-gate is the output of an AND-gate representing the conjunction of U_i and the negation of its specific inhibitory mechanism I_i. The inhibitors $I_1, \ldots, I_n$ represent exceptions or abnormalities that interfere with the normal relationship between $\boldsymbol{U}$ and X. We use q_i to represent the probability that the i-th inhibitor is active. Assume all inputs are 0 except $U_i = 1$. X will only be 1 if and only if the inhibitor I_i associated with U_i remains inactive. That is,

$$P(X = 1 | U_i = 1, U_k = 0 \; k \neq i) = 1 - q_i. \tag{6.12}$$

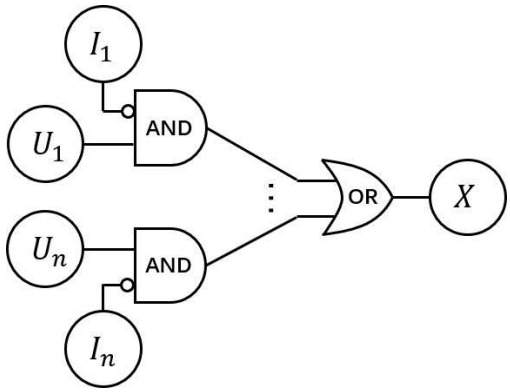


Figure 6.9: The noisy OR-gate

Therefore, $c_i = 1 - q_i$ represents the degree to which the single cause $U_i = 1$ can endorse the consequent event $X = 1$. Let

$$\boldsymbol{u} = (u_1, u_2, \ldots, u_n) \quad u_i \in \{0, 1\} \tag{6.13}$$

represent any assignment of values to parent set $\boldsymbol{U}$. Note that both $\boldsymbol{u}$ and $\boldsymbol{U}$ are vectors since X could have multiple parents. Let $T_u = \{i : U_i = 1\}$ represent the subset of parents that are 1. In the "noisy-OR" model, a link matrix $P(x|\boldsymbol{u})$ is used to relate X to its parent set $\boldsymbol{U}$ and can be written as

$$P(x|\boldsymbol{u}) = \begin{cases} \prod_{i \in T_u} q_i & \text{if} \quad x = 0 \\ 1 - \prod_{i \in T_u} q_i & \text{if} \quad x = 1. \end{cases} \tag{6.14}$$

Having the link matrix $P(x|\boldsymbol{u})$, we can follow similar belief propagation algorithm described in Section 6.3.3. Assume X has m children, $Y_1, Y_2, \ldots, Y_m$. As it is shown in Figure 6.10, the local belief updating at X can be also executed by three steps in any order.

Belief Propagation Algorithm in "Noisy-OR" Model

Step 1 — Belief updating: Node X updates its belief measure based on the $\pi_{1X}, \ldots, \pi_{nX}$ from its parents and the $\lambda_{Y_1}(x), \ldots, \lambda_{Y_m}(x)$ from its children:

$$BEL(x) = \begin{cases} \alpha\lambda_0 \prod_i (1 - c_i\pi_{iX}) & \text{if} \quad x = 0 \\ \alpha\lambda_1 \Big[1 - \prod_i (1 - c_i\pi_{iX})\Big] & \text{if} \quad x = 1, \end{cases} \tag{6.15}$$

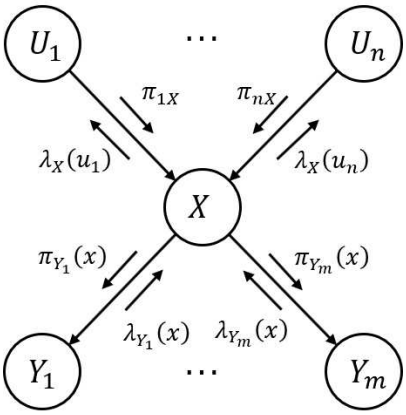


Figure 6.10: Belief propagation in causal polytree

where

$$\lambda(x) = \prod_j \lambda_{Y_j}(x) = \begin{cases} \lambda_0 & \text{if} \quad x = 0 \\ \lambda_1 & \text{if} \quad x = 1 \end{cases}, \tag{6.16}$$

$$\pi_{iX} = P(u_i = 1), \tag{6.17}$$

and α is a normalizing constant rendering $\sum_x BEL(x) = 1$.

Step 2 — Bottom-up propagation: Node X computes new $\lambda_X(u_i)$ messages and sends them to its parents $\boldsymbol{U}$. The new $\lambda_X(u_i)$ message for its i-th parent U_i is calculates as

$$\lambda_X(u_i) = \begin{cases} \beta\big[\lambda_0 q_i \Pi_i' + \lambda_1(1 - q_i\Pi_i')\big] & \text{if} \quad u_i = 1 \\ \beta\big[\lambda_0 \Pi_i' + \lambda_1(1 - \Pi_i')\big] & \text{if} \quad u_i = 0, \end{cases} \tag{6.18}$$

where

$$\Pi_i' = \prod_{k \neq i}(1 - c_k \pi_{kX}), \tag{6.19}$$

and β is a normalizing constant.

Step 3 — Top-down propagation: Node X computes new $\pi_{Y_j}(x)$ messages and sends them to its children. The new $\pi_{Y_j}(x)$ message for its j-th child Y_j is calculated as

$$\pi_{Y_j}(x) = \alpha \frac{BEL(x)}{\lambda_{Y_j}(x)}. \tag{6.20}$$

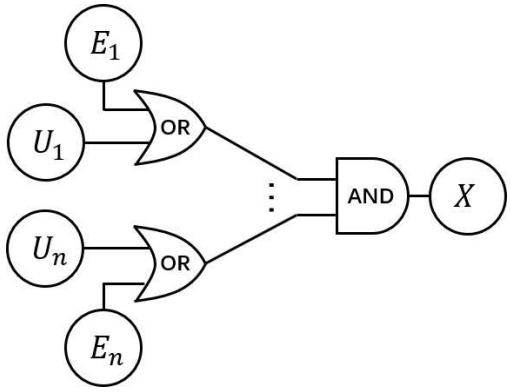


Figure 6.11: The noisy AND-gate

The "noisy-AND" Model

The "noisy-AND" model [102] is based on the noisy AND-gate structure shown in Figure 6.11. The value of a node X with n parents $\boldsymbol{U} = \{U_1, U_2, \ldots, U_n\}$ can be seen as the output of a logical AND-gate. Each input to the AND-gate is the output of an OR-gate representing the conjunction of U_i and the its specific enabling mechanism E_i. We use q_i to represent the probability that the i-th enabler is active. Assume all inputs are 1 except $U_i = 0$. X will be 0 if and only if the enabler E_i associated with U_i remains inactive. That is,

$$P(X = 1|U_i = 0, U_k = 1 \; k \neq i) = q_i. \tag{6.21}$$

Let $c_i = 1 - q_i$ and use $F_u = \{i : U_i = 0\}$ to represent the subset of parents that are 0. The link matrix $P(x|\boldsymbol{u})$ can be written as

$$P(x|\boldsymbol{u}) = \begin{cases} 1 - \prod_{i \in F_u} q_i & \text{if} \quad x = 0 \\ \prod_{i \in F_u} q_i & \text{if} \quad x = 1. \end{cases} \tag{6.22}$$

Assume X has m children, Y_1, Y_2, ..., Y_m. The three steps of local belief updating at X are listed as follows.

Belief Propagation Algorithm in "Noisy-AND" Model

Step 1 — Belief updating: Node X updates its belief measure based on the π_{1X}, ..., π_{nX} from its parents and the $\lambda_{Y_1}(x)$, ..., $\lambda_{Y_m}(x)$ from its children:

$$BEL(x) = \begin{cases} \alpha\lambda_0 \Big\{ 1 - \prod_i \big[1 - c_i(1 - \pi_{iX}) \big] \Big\} & \text{if} \quad x = 0 \\ \alpha\lambda_1 \prod_i \big[1 - c_i(1 - \pi_{iX}) \big] & \text{if} \quad x = 1, \end{cases} \tag{6.23}$$

where

$$\lambda(x) = \prod_j \lambda_{Y_j}(x) = \begin{cases} \lambda_0 & \text{if} \quad x = 0 \\ \lambda_1 & \text{if} \quad x = 1 \end{cases}, \tag{6.24}$$

$$\pi_{iX} = P(u_i = 1), \tag{6.25}$$

and α is a normalizing constant rendering $\sum_x BEL(x) = 1$.

Step 2 — Bottom-up propagation: Node X computes new $\lambda_X(u_i)$ messages and sends them to its parents $\boldsymbol{U}$. The new $\lambda_X(u_i)$ message for its i-th parent U_i is calculates as

$$\lambda_X(u_i) = \begin{cases} \beta\big[\lambda_0(1 - \Pi_i') + \lambda_1 \Pi_i'\big] & \text{if} \quad u_i = 1 \\ \beta\big[\lambda_0(1 - q_i\Pi_i') + \lambda_1 q_i \Pi_i'\big] & \text{if} \quad u_i = 0, \end{cases} \tag{6.26}$$

where

$$\Pi_i' = \prod_{k \neq i} \big[1 - c_k(1 - \pi_{kX})\big], \tag{6.27}$$

and β is a normalizing constant.

Step 3 — Top-down propagation: Node X computes new π messages and send them to its children. The new $\pi_{Y_j}(x)$ message for its j-th child Y_j is calculates as

$$\pi_{Y_j}(x) = \alpha \frac{BEL(x)}{\lambda_{Y_j}(x)}. \tag{6.28}$$

6.4 DESIGN OVERVIEW

As we mentioned in Section 6.1, CAPTAR resides in the control center of the SCADA network and its inputs are meta-alerts sent by EDMAND at the edge of the network. In this section, we present a design overview of CAPTAR. The main architecture of CAPTAR is shown in Figure 6.12. CAPTAR consists of 4 components: (1)*Meta-alert Database*, (2)*Attack Template Database*, (3)*Alert Correlator*, (4)*Causal Reasoning Engine*.

The meta-alert database is used to store the meta-alerts from EDMAND. These meta-alerts serve as evidence to our causal reasoning of anomalies. The attack template database stores the potential attack templates which are causal polytrees created by domain experts. These attack templates are Bayesian networks mentioned in Section 6.3.1 and contain the "noisy-OR" and "noisy-AND" models mentioned in Section 6.3.4. They represent the prior domain knowledge we have on potential attack plans and are used as the underlying Bayesian

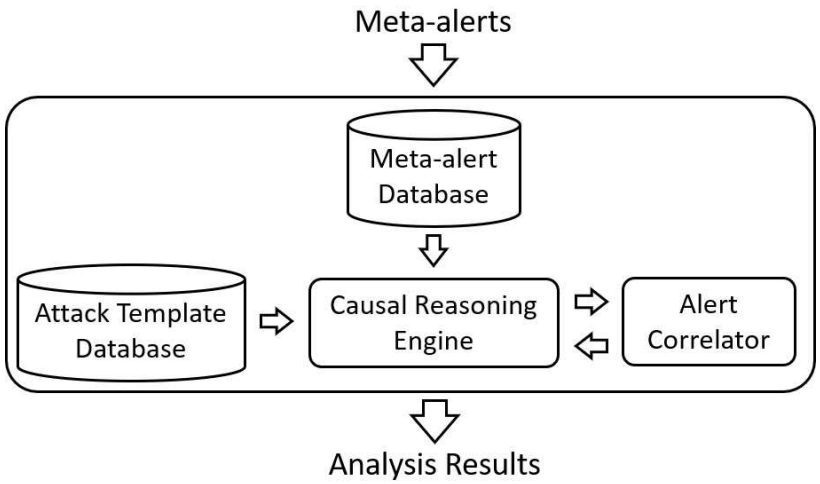


Figure 6.12: CAPTAR architecture

networks for the belief propagation mentioned in Section 6.3.3. One important step to reason about the anomalies is to match meta-alerts to nodes (attack steps) in our attack templates. And to do that, we need to evaluate whether one meta-alert is correlated with other meta-alerts. The alert correlator takes two meta-alerts as inputs and outputs a correlation score which is used to decide whether the two input meta-alerts are correlated or not. The core component of CAPTAR is the causal reasoning engine which interacts with all other three components. When the causal reasoning engine is started, it fetches copies of the attack templates in the database and conducts alert matching as well as belief propagation on them. The meta-alerts are retrieved from the meta-alert database and the alert matching is done using the alert correlator. Whenever the belief of an attack is high enough, the engine outputs the causal polytree corresponding to that attack with matched alerts. The operator can further analyze the believes and matched alerts in the causal polytree to understand each step of the attack.

In the following subsections, we will introduce the meta-alert, the attack template, the alert correlator, and the causal reasoning engine in more detail.

6.4.1 Meta-alert

Meta-alerts are generated by EDMAND in Chapter 5 and sent to the control center where CAPTAR resides. Each meta-alert is the aggregation of similar alerts and the aggregation rules are mentioned in Table 5.4. Each meta-alert has several fields which are listed in Table 6.3. The fields that will be used in CAPTAR are alert id, alert type, index field, timestamp, confidence score. Alert id is a string that is unique for each meta-alert. The received

meta-alerts from EDMAND will be first stored in the meta-alert database (implemented by MongoDB). The alert id serves as the key to locate and retrieve the meta-alert from the database. Alert type is a name that briefly describes the meta-alert. The current prototype of EDMAND generates 24 types of alerts from the transport, operation, and content levels. A complete list of the alert types is shown in Table 6.4. For simplicity reason, we assign an alert type index to each alert type and we will use the index to represent the corresponding alert type. Index field of the meta-alert contains additional information that helps to describe the meta-alert, such as IP addresses, protocol, service, etc. This field is later used by the alert correlator to correlate meta-alerts. Timestamp field simply contains a pair of timestamps (start time, end time). They are the timestamps of the earliest and the latest alerts that have been aggregated to the meta-alert. Confidence score field in the meta-alert represents the confidence that the meta-alert is an anomaly indeed. It is the maximum of the confidence scores of all the aggregated alerts for this meta-alert. As we mentioned in Section 5.3.2, the confidence score (CS) for alert is calculated by $CS = MA \times AS$, where MA is the model accuracy and AS is the anomaly score. As stated in Section 6.3.3, if meta-alerts are matched to a node X in our causal polytree, an auxiliary child node $\widetilde{X}$ is added to X. The confidence scores of the matched meta-alerts are used to calculate the diagnostic support message $\lambda_{\widetilde{X}}(x)$ that $\widetilde{X}$ provides to X. The way to calculate $\lambda_{\widetilde{X}}(x)$ will be introduced in Section 6.4.2.

Alert Field	Description
alert id	a unique id for retrieving a meta-alert from the database
alert type	a name that describes the meta-alert (see Table 6.4)
index field	a set of auxiliary information that helps to describe the meta-alert (refer to Table 5.1)
timestamp	a start time and an end time for the meta-alert
confidence score	the confidence that the meta-alert represent an anomaly
statistical fields	more detailed information about the last alert aggregated
anomaly data	
count	number of alerts aggregated by the meta-alert

Table 6.3: Meta-alert fields and description

6.4.2 Attack Template

As we mentioned in Section 6.3, we utilize causal polytrees to reason about anomalies in SCADA networks. We call these special causal polytrees attack templates and use ATs to represent them. Attack templates represent and store the prior domain knowledge we

Index	Alert Type	Alert Level
0	PACKET_IAT	Transport: Packet
1	PACKET_BYTES	
2	NEW_ORIG	
3	NEW_RESP	
4	NEW_PROTOCOL	
5	NEW_SERVICE	
6	PACKET_AB_TOO_MANY	Transport: Flow
7	PACKET_AB_TOO_FEW	
8	PACKET_BA_TOO_MANY	
9	PACKET_BA_TOO_FEW	
10	MEAN_BYTES_AB_TOO_LARGE	
11	MEAN_BYTES_AB_TOO_SMALL	
12	MEAN_BYTES_BA_TOO_LARGE	
13	MEAN_BYTES_BA_TOO_SMALL	
14	OPERATION_TOO_LATE	Operation
15	OPERATION_TOO_EARLY	
16	OPERATION_MISSING	
17	INVALID_FUNCTION_CODE	
18	RESPONSE_FROM_ORIG	
19	REQUEST_FROM_RESP	
20	NEW_OPERATION	
21	BINARY_FAULT	Content
22	ANALOG_TOO_LARGE	
23	ANALOG_TOO_SMALL	

Table 6.4: Alert type

have for attacks. When an attack is launched, the triggered meta-alerts from EDMAND are matched to the corresponding attack template and the belief propagation mentioned in Section 6.3.3 is conducted on it. An example attack template for the data integrity attack is shown in Figure 6.13. Each node X in an attack template AT is an attack step with zero, one, or multiple parents and children. Each parent represents a prior cause attack step that can lead to the current one and each child represents a posterior consequence attack step that the current one can lead to. If there are multiple parents, they follow either the "noisy-OR" or the "noisy-AND" model in Section 6.3.4. The prior probability at each node, the probabilities q_is of the inhibitory or enabling mechanisms in "noisy-OR" and "noisy-AND" models are all specified by domain experts (e.g. power grid/SCADA security administrator) when the attack template is created. Also, each attack template AT contains one or more sink nodes (shaded node in Figure 6.13). Denote the set of sink nodes as $\boldsymbol{S}_{AT}$. Nodes in $\boldsymbol{S}_{AT}$ represent the final targets of the entire attack and we call them consequence nodes.

Each consequence node has domain knowledge associated with such as attack consequence, severity, and potential countermeasure.

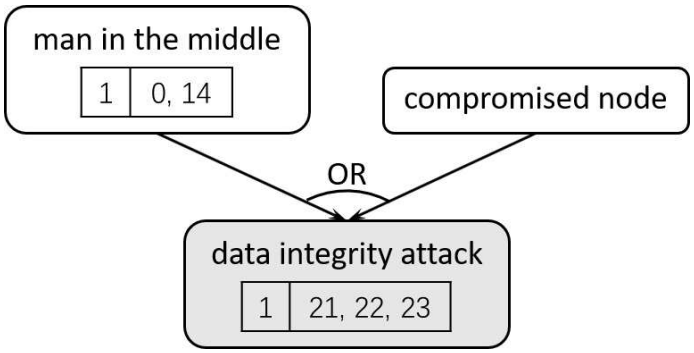


Figure 6.13: An example of an attack template (causal polytree) using the "noisy-OR" model

Each attack step (each node) has two binary states: not exist (0) and exist (1). However, the attack steps cannot be observed directly. We can only infer the existence of each attack step by the alerts it triggers in EDMAND. Each attack step could trigger meta-alerts that belong to multiple[1] alert types mentioned in Section 6.4.1. Multiple meta-alerts can match to one alert type of an attack step. As we mentioned in Section 6.3.3, these alerts are treated as evidence to each attack step node. We create a structure, called alert unit table, to store the matched meta-alerts at each attack step. An example of the alert unit table is shown in Table 6.5. Each row in the table is an alert unit (AU), which represents one proportion of evidence. Let us assume there are k alert units in the table. Each alert unit AU_i consists of a weight w_i and a list of alert types $A_{i1}, A_{i2}, \ldots, A_{in_i}$, where n_i is the number of alert types in AU_i. Therefore, $AU_i = \{w_i, A_{i1}, A_{i2}, \ldots, A_{in_i}\}$. As we mentioned in Section 6.4.1, our current prototype of EDMAND can generate 24 types of alerts. $A_{i1}, A_{i2}, \ldots, A_{in_i}$ are represented by the alert type indexes in Table 6.4 for simplicity reason. w_i represents how much the observation of one or more of the following alert types $A_{i1}, A_{i2}, \ldots, A_{in_i}$ can prove the existence of the attack step and $\sum_i w_i = 1$. Alert types in the same alert unit express the same aspect of the attack step. Each alert type A_{ij} in the alert unit table can contain multiple meta-alerts from EDMAND of the same corresponding alert type. For example, in the "data integrity attack" attack step in Figure 6.13, the alert unit table contains one alert unit $\{1, 21, 22, 23\}$. Since there is just one alert unit, its weight is 1. The three alert types are 21, 22, and 23, which represent `BINARY_FAULT`, `ANALOG_TOO_LARGE`, and `ANALOG_TOO_SMALL`. These three types of content-level meta-alerts all represent the actual tampering of the measurement data and are therefore included in the same alert unit.

[1]It is also possible that one attack step triggers no alerts in EDMAND. In this case, we can only infer the existence of this attack step by the existence of its parents and children.

Alert Unit	Weight	Alert Types
AU_1	w_1	$A_{11}, A_{12}, \ldots, A_{1n_1}$
AU_2	w_2	$A_{21}, A_{22}, \ldots, A_{2n_2}$
$\vdots$	$\vdots$	$\vdots$
AU_k	w_k	$A_{k1}, A_{k2}, \ldots, A_{kn_k}$

Table 6.5: Alert unit table for each attack step

As we mentioned in Section 6.3.3, if meta-alerts are matched to a node X and stored in its alert unit table, an auxiliary child node $\widetilde{X}$ is added to X. The confidence scores of the matched meta-alerts are used to calculate the diagnostic support message $\lambda_{\widetilde{X}}(x)$ that $\widetilde{X}$ provides to X. To calculate $\lambda_{\widetilde{X}}(x)$, we utilize the confidence score of each meta-alert mentioned in Section 6.4.1. For each alert type A_{ij} in the alert unit table, we assume there are m_{ij} meta-alerts $a_{ij1}, a_{ij2}, \ldots, a_{ijm_{ij}}$ matched to it (the matching mechanism will be described in Section 6.4.4). The confidence scores of them are $CS(a_{ij1}), CS(a_{ij2}), \ldots, CS(a_{ijm_{ij}})$. Let $CS(A_{ij})$ be the confidence score of the alert type A_{ij} and it is calculated as

$$CS(A_{ij}) = \begin{cases} \dfrac{\prod_{l=1}^{m_{ij}} CS(a_{ijl})}{\prod_{l=1}^{m_{ij}} CS(a_{ijl}) + \prod_{l=1}^{m_{ij}} (1 - CS(a_{ijl}))} & \text{if} \quad m_{ij} > 0 \\ P_{miss} & \text{if} \quad m_{ij} = 0, \end{cases} \tag{6.29}$$

where P_{miss} is a probability of missing meta-alerts and can be predefined by experience or calculated if training data is available. After we have confidence score calculated for every alert type in one alert unit AU_i, we can write the confidence score of the alert unit $CS(AU_i)$ as

$$CS(AU_i) = \max_{j=1}^{n_i} CS(A_{ij}). \tag{6.30}$$

The final total confidence score of the attack step CS_{total} is calculated by

$$CS_{total} = \sum_{i=1}^{k} w_i CS(AU_i). \tag{6.31}$$

The diagnostic support $\lambda_{\widetilde{X}}(x)$ provided by all the matched alerts to the attack step X is written as

$$\lambda_{\widetilde{X}}(x) = \begin{cases} 1 - CS_{total} & \text{if} \quad x = 0 \\ CS_{total} & \text{if} \quad x = 1 \end{cases}. \tag{6.32}$$

Attack templates are created by domain experts and stored in the attack template database before we start the anomaly reasoning. At the beginning of the reasoning, the causal reason-

ing engine will fetch copies of the original attack templates and create an attack template set $\boldsymbol{ATS}$. Then the engine conducts alert matching as well as the belief propagation mentioned in Section 6.3.3 on them. Each attack template AT in $\boldsymbol{ATS}$ originates from one attack template in the database. And multiple attack templates in $\boldsymbol{ATS}$ could correspond to the same attack (same attack template in the database). For each attack step X in an attack template AT, $BEL_X(1)$ represents the probability of existence of this attack step. The way to calculate $BEL_X(1)$ is introduced in Section 6.3. Since consequence nodes in $\boldsymbol{S}_{AT}$ stand for final targets of the entire attack represented by AT, the maximum probability of existence of all consequence nodes in AT, denoted by $BEL_{max}(AT)$, can represent the inferred success possibility of the attack and is calculated as

$$BEL_{max}(AT) = \max_{X \in \boldsymbol{S}_{AT}} BEL_X(1). \tag{6.33}$$

6.4.3 Alert Correlator

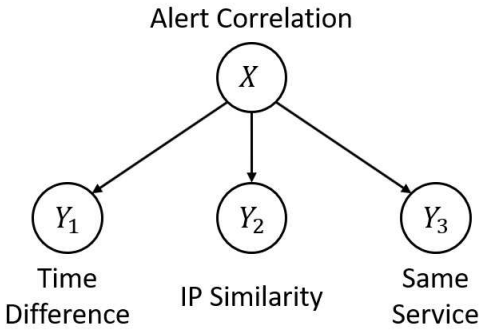


Figure 6.14: Alert correlation model

CAPTAR's anomaly reasoning consists of meta-alert matching and belief propagation. Meta-alert matching is the process of matching meta-alerts to attack steps (in attack templates) that trigger them. And the most important step of alert matching is to decide whether two meta-alerts are correlated or not. Therefore, the alert correlator is designed for this purpose. The alert correlator is a naive Bayes classifier whose graphical representation is a Bayesian network in Figure 6.14 with one root node X and three leaf nodes Y_1, Y_2, and Y_3. The root node X represents the hypothesis that "the two input meta-alerts are correlated" and has two states: "yes" (1) and "no" (0). Each leaf node Y_j ($j \in \{1, 2, 3\}$) stands for one type of observable evidence that helps to evaluate the hypothesis and has several discrete states. Depending on whether two meta-alerts are correlated or not, the distribution of states at the evidence nodes will be different. Therefore, based on the observed states at

the evidence nodes, one can infer the probability that two meta-alerts are correlated. We consider three kinds of observable evidence while correlating two meta-alerts: *time difference* (Y_1), *IP similarity* (Y_2), and whether they share the *same service* (Y_3).

- *Time difference:* The state of Y_1 depends on the closeness in the time axis of the two meta-alerts and we use T_{diff} to represent that. As we described in Section 6.4.1, each meta-alert from EDMAND has a start time and an end time. If the two meta-alerts overlap, we assign 0 to T_{diff}. Otherwise, we calculate T_{diff} as the difference between the end time of the earlier meta-alert and the start time of the latter one. Y_1 has four corresponding states according to T_{diff}:

$$Y_1 = \begin{cases} 0 & \text{if} \quad T_{diff} \leq 60\text{seconds} \\ 1 & \text{if} \quad 60\text{seconds} < T_{diff} \leq 1\text{hour} \\ 2 & \text{if} \quad 1\text{hour} < T_{diff} \leq 1\text{day} \\ 3 & \text{if} \quad T_{diff} > 1\text{day} \end{cases}. \tag{6.34}$$

- *IP similarity:* The state of Y_2 depends on the similarity of IP addresses related to the two meta-alerts. Each meta-alert could have one or two related IP addresses. Content-level alerts have one measure source IP while transport and operation level alerts have two IPs for originator and responder. For every pair of IP addresses (IP_a, IP_b), where IP_a relates to one input meta-alert and IP_b relates to the other, we calculate the similarity of them as follows:

$$SIM(IP_a, IP_b) = \begin{cases} 3 & \text{if} \quad IP_a \text{ and } IP_b \text{ are exactly the same} \\ 2 & \text{if} \quad IP_a \text{ and } IP_b \text{ are not the same but within the same 8-bit block} \\ 1 & \text{if} \quad IP_a \text{ and } IP_b \text{ are not within the same 8-bit block but within the same 16-bit block} \\ 0 & \text{if} \quad IP_a \text{ and } IP_b \text{ are not within the same 16-bit block} \end{cases} \tag{6.35}$$

  The maximum similarity of all such IP pairs is selected as the state of Y_2. Therefore, Y_2 has four states of $\{0, 1, 2, 3\}$ and $Y_2 = \max_{(IP_a, IP_b)} SIM(IP_a, IP_b)$.

- *Same service:* Y_3 evaluates whether the two meta-alerts share the same service (i.e., the same industrial control protocol). There are two states of Y_3: "yes" (1) and "no"

(0). A "no" is also specified if any of the input meta-alerts does not have a related service.

Let x ($x \in \{0, 1\}$) represent the state of the root node in Figure 6.14. Let y_j ($j \in \{1, 2, 3\}$) represent the state at each leaf node Y_j and $\widehat{y}_j$ represent the already observed state. There is a conditional probability table (CPT) at each leaf node Y_j which relates Y_j to X. As we stated in Section 6.3.3, each entry $P(y_j|x)$ in the table defines the probability of state y_j of node Y_j given certain state x of node X. Since X is a root node with no parent, we set $pi(x)$ to be the prior probability $P(x)$ according to the boundary condition mentioned in Section 6.3.3. $P(x)$ varies depending on the alert types of the two input meta-alerts. There is a predefined prior probability for each pair of alert types based on domain knowledge. And since the state of Y_j is already observed as $\widehat{y}_j$, we have

$$\lambda(y_j) = \begin{cases} 1 & \text{if} \quad y_j = \widehat{y}_j \\ 0 & \text{otherwise.} \end{cases} \tag{6.36}$$

According to the bottom-up propagation step in the belief propagation, the diagnostic support provided by Y_k to X is $\lambda_{Y_j}(x) = \sum_{y_j} \lambda(y_j) P(y_j|x) = P(\widehat{y}_j|x)$. Therefore, the belief at root X can be calculated as

$$BEL(x) = \alpha\lambda(x)\pi(x) = \alpha\pi(x) \prod_{j=1}^{3} \lambda_{Y_j}(x) = \alpha P(x) \prod_{j=1}^{3} P(\widehat{y}_j|x), \tag{6.37}$$

where α is a normalizing factor rendering $\sum_x BEL(x) = 1$. We say two meta-alerts are correlated if $BEL(1) > 0.5$ for X. Let a and b be the two input meta-alerts for the alert correlator. We define the CORRELATE procedure of the alert correlator as follows:

$$\text{CORRELATE}(a, b) = \begin{cases} BEL(1) & \text{if} \quad BEL(1) > 0.5 \\ -1 & \text{otherwise,} \end{cases} \tag{6.38}$$

6.4.4 Causal Reasoning Engine

The causal reasoning engine is the core component of CAPTAR and it interacts with all other three components. When the causal reasoning engine starts, it fetches copies of attack templates ATs from the attack template database and creates an attack template set ATS. Then it runs an anomaly reasoning algorithm to perform alert matching and belief propagation on the attack templates in the attack template set. The meta-alerts used in

the alert matching are retrieved from the meta-alert database and the alert correlator is also used to correlate meta-alerts during the matching process. The belief propagation is introduced in Section 6.3.

The anomaly reasoning algorithm is shown in Algorithm 6.1. The AnalyzeAlert procedure in this algorithm is called whenever CAPTAR receives a new meta-alert or an update to an existing alert. The procedure takes the meta-alert a and the current attack template set $\boldsymbol{ATS}$ in the causal reasoning engine as inputs. The output is a new attack template set $\boldsymbol{ATS}_{new}$ with the meta-alert a matched to some of the attack templates inside and belief propagation performed. The procedure has two cases. If a is an update to an existing meta-alert, then some attack templates in $\boldsymbol{ATS}$ might already have a matched. For each AT of those attack templates, the algorithm gets the node X in AT that a is matched to. Since the meta-alert is updated, the procedure recalculates the total confidence score CS_{total} (presented in Section 6.4.2) of X. The diagnostic support $\lambda_{\tilde{x}}(x)$ from all the matched alerts is also recalculated. Since the evidence contained at X changes, a belief propagation in AT from node X is initiated. In this case, the $\boldsymbol{ATS}$ with the updated attack templates are directly assigned to $\boldsymbol{ATS}_{new}$ for output. If a is a newly detected meta-alert, the algorithm iterates over the entire set $\boldsymbol{ATS}$. For each attack template AT in $\boldsymbol{ATS}$, it matches the meta-alert a to nodes in AT and performs a belief propagation if there is a successful match. This process is included in the procedure called MatchAlert. This procedure takes a and AT as inputs and outputs a set of attack templates $\boldsymbol{ATS}_{match}$. The attack templates in $\boldsymbol{ATS}_{match}$ are copies of AT with a matched and belief propagation performed. Since it is possible that a can match to multiple nodes in AT, $\boldsymbol{ATS}_{match}$ could contain multiple copies. If a cannot be matched to AT, $\boldsymbol{ATS}_{match}$ will just contain the original AT. After we get $\boldsymbol{ATS}_{match}$ from MatchAlert(a, AT), the attack templates in $\boldsymbol{ATS}_{match}$ are all added to $\boldsymbol{ATS}_{new}$. After each run of the algorithm, namely each call of procedureAnalyzeAlert, the attack template set $\boldsymbol{ATS}$ in the causal reasoning engine is replaced by $\boldsymbol{ATS}_{new}$. The engine then checks $BEL_{max}(AT)$ (defined in Section 6.4.2) of every attack template AT in the new attack template set. If it finds $BEL_{max}(AT) > \theta_{BEL}$ for any AT, it will output that attack template AT for operator's further analysis. Here θ_{BEL} is a predefined threshold and we use $\theta_{BEL} = 0.8$ for our CAPTAR prototype.

Before we introduce more details of the MatchAlert procedure, there are one concept and another procedure we need to describe first. The concept is called *happens before* and the procedure's name is FindCorrelation. *Happens before* is a relationship between two meta-alerts. We say meta-alert a *happens before* meta-alert b if the start time of a is at least T_{hb} earlier than the start time of b, where $T_{hb} = 10s$ is a predefined threshold. The procedure FindCorrelation is shown in Algorithm 6.2. It takes a meta-alert a

Algorithm 6.1 Anomaly Reasoning Algorithm

```
Input:
    a - meta-alert to be analyzed
    ATS - attack template set
Output:
    ATS_new - new attack template set
procedure AnalyzeAlert(a, ATS)
    ATS_new ← ∅
    if a is an update of an existing meta-alert then
        for each AT in ATS that has a as a matched alert do
            recalculate CS_total and λ_x̃(x) of the matched node X
            start a new belief propagation in AT from node X
        end for
        ATS_new ← ATS
    else
        for each AT in ATS do
            ATS_match ← MatchAlert(a, AT)
            add ATS_match to ATS_new
        end for
    end if
    return ATS_new
end procedure
```

and a node X in the attack template as inputs and outputs a correlation score Cor_{max}. The objective of this procedure is to find whether the given node, its parents and children have any matched alert that correlates with the given alert. The procedure does so by iterating through every matched alert b of X, parents of X and children of X. For each b, it calls the alert correlator and uses the CORRELATE procedure to correlate a and b. The maximum result from CORRELATE(a, b) is stored in Cor_{max}. If any correlation is found, Cor_{max} contains the highest correlation score. Otherwise, $Cor_{max} = 0$. There are two exceptions while correlating alerts from parents and children. For any matched alert b of X's parents, there is a conflict if a *happens before* b. a is to be matched to X and X's parents are attack steps that should lead to X. If there is an attack, the attack steps represented by X's parents should be launched before X. That means a could not *happens before* b. Therefore, a should not be matched to X and the procedure outputs -1 in this case. For any matched alert b of X's children, the procedure outputs -1 if b *happens before* a for similar reasons.

After describing the *happens before* concept and the FINDCORRELATION procedure, we can start looking at the MATCHALERT procedure which is shown in Algorithm 6.3. It takes a meta-alert a and an attack template AT as inputs and outputs a set $\boldsymbol{ATS}_{match}$ containing

Algorithm 6.2 Find Correlation Procedure

```
Input:
    a - meta-alert to find correlation with
    X - a node in the attack template whose alert unit table contains the alert type of a
Output:
    Cor_max - maximum correlation
procedure FindCorrelation(a, X)
    Cor_max ← 0
    for each matched alert b of X do
        Cor_max ← max(Cor_max, Correlate(a, b))
    end for
    for each parent U of X do
        for each matched alert b of U do
            if a happens before b then
                return −1
            end if
            Cor_max ← max(Cor_max, Correlate(a, b))
        end for
    end for
    for each child Y of X do
        for each matched alert b of Y do
            if b happens before a then
                return −1
            end if
            Cor_max ← max(Cor_max, Correlate(a, b))
        end for
    end for
    return Cor_max
end procedure
```

attack templates generated after matching. The objective of this procedure is to try to match meta-alert a to the attack template AT. It iterates over every node X in AT whose alert unit table contains alert type of a. It calls the procedure FindCorrelation to correlate a with X. If the result is greater than 0, it means a finds correlation in X. If the result is 0, it means a finds no correlation in X but it can be matched to X. We add X to a potential node set $\boldsymbol{X}_{pot}$. If the result is less than 0, it means a could not be matched to X due to conflicts. If a finds correlation in any node in AT, this means we have good reason to believe a is triggered by the attack represented by the current attack template AT. Therefore, we match a to the node X_{cor} with the highest correlation score and start a belief propagation from X_{cor}. In this case, the output will be a set containing only the updated AT with a matched. If a finds no correlation in AT but $\boldsymbol{X}_{pot}$ is not empty, this means there is no proof

that a is triggered by AT but there are attack steps in AT that could potentially trigger a and the attack steps are included in $\boldsymbol{X}_{pot}$. Therefore, the procedure iterates over $\boldsymbol{X}_{pot}$ explores every possibility. For every node X in $\boldsymbol{X}_{pot}$, it creates a new copy AT_{match} of AT. Note that this copy contains not only the nodes of AT but also all already matched alerts of AT. It then matches a to X's counterpart in AT_{match} and starts a belief propagation in AT_{match} from that node. By doing this, the procedure takes every potential match of a in AT into consideration and the final output will contain the original AT as well as all updated copies of it. Finally, if there is no node in AT that a could match to, the output will just contain the original AT.

Algorithm 6.3 Match Alert Procedure

Input:
a - meta-alert to be matched
AT - attack template
Output:
$\boldsymbol{ATS}_{match}$ - attack template set after matching
procedure MatchAlert(a, AT)
$\boldsymbol{ATS}_{match} \leftarrow \{AT\}$, $X_{cor} \leftarrow None$, $\boldsymbol{X}_{pot} \leftarrow \emptyset$, $Cor_{max} \leftarrow 0$
for each node X in AT whose alert unit table contains alert type of a **do**
$Cor \leftarrow$ FindCorrelation(a, X)
if $Cor > 0$ **then**
if $Cor > Cor_{max}$ **then**
$Cor_{max} \leftarrow Cor$, $X_{cor} \leftarrow X$
end if
else if $Cor = 0$ **then**
add X to $\boldsymbol{X}_{pot}$
end if
end for
if X_{cor} is not $None$ **then**
match a to X_{cor} and start the belief propagation of AT from X_{cor}
else
for each node X in $\boldsymbol{X}_{pot}$ **do**
$AT_{match} \leftarrow$ copy of AT
match a to X in AT_{match} and start a belief propagation of AT_{match} from X
add AT_{match} to $\boldsymbol{ATS}_{match}$
end for
end if
return $\boldsymbol{ATS}_{match}$
end procedure

In the description of the MatchAlert procedure, we mentioned that the procedure will explore every potential match of a and create multiple copies of the original attack template

AT if no exact match can be found. This will increase the number of attack templates in the attack template set $\boldsymbol{ATS}$. To prevent the number of attack templates from exploding, we set a maximum limit K for the number of attack templates to keep for each kind of attack. Attack templates with lower $BEL_{max}(AT)$ will be dropped when the number exceeds the limit. Also, attack templates will also be dropped from the set if they have not been updated for a long time.

The attack templates, output by the causal reasoning engine, represent attacks of high probability of existence in the SCADA network. The operators can not only understand the origin of the attacks by examining the belief of each attack step and the corresponding alerts, but also evaluate the attack consequences and take countermeasures by utilizing the domain knowledge contained in the consequence nodes.

Example Run for the Anomaly Reasoning Algorithm

We use an example to better illustrate the anomaly reasoning algorithm. Consider the attack template AT in Figure 6.15. Let us assume this is the only attack template in the database. At the beginning of the anomaly reasoning, the causal reasoning engine fetches AT from the database and creates the attack template set $\boldsymbol{ATS} = \{AT\}$ as shown in Figure 6.16. Now the attacker first launched a man-in-the-middle attack. CAPTAR receives an `OPERATION_TOO_LATE` meta-alert a_1 from EDMAND. This meta-alert a_1 is first stored in the meta-alert database and then fed into the causal reasoning engine. Upon receiving this meta-alert a_1, the engine calls the anomaly reasoning algorithm. It finds that a_1 is a new meta-alert, so the procedure MatchAlert is called to match a_1 to the only attack template AT in $\boldsymbol{ATS}$. The MatchAlert procedure finds that node X_1 is the only node whose alert unit tables contains alert type `OPERATION_TOO_LATE`. Therefore, it calls the procedure FindCorrelation with a_1 and X_1 as inputs. The procedure FindCorrelation tries to find any meta-alert in X_1 and X_3 that correlates with meta-alert a_1. However, since X_1 and X_3 have no matched alert yet, the procedure finds no correlation and returns 0 in this case. Since no correlation of a_1 in the attack template AT is found and X_1 is the only node that a_1 can match to, a copy $\widehat{AT}$ of the attack template AT is created, and the meta-alert a_1 is matched to $\widehat{X_1}$ in the copy $\widehat{AT}$. A belief propagation is performed on $\widehat{AT}$. The MatchAlert procedure returns both AT and $\widehat{AT}$. Finally, both AT and $\widehat{AT}$ are added to $\boldsymbol{ATS}_{new}$ to replace $\boldsymbol{ATS}$.

Now the attacker intercepts and tampers with some binary data. CAPTAR receives a meta-alert a_2 with alert type of `BINARY_FAULT` from EDMAND as shown in Figure 6.17. a_2 is first stored in the meta-alert database and then forwarded to the causal reasoning

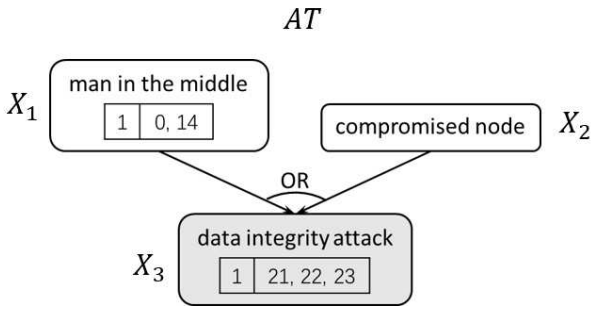


Figure 6.15: Example attack template AT

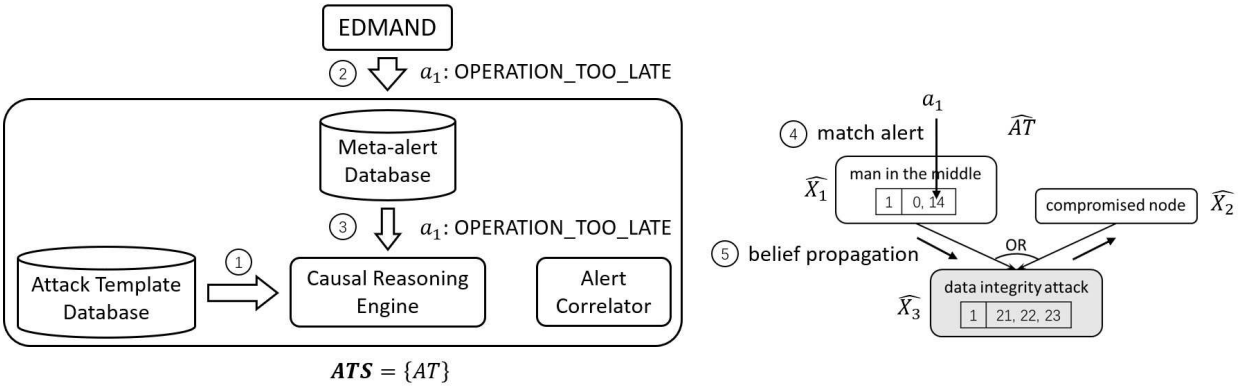


Figure 6.16: Algorithm run upon receiving meta-alert a_1

engine. The anomaly reasoning algorithm again finds that a_2 is a new meta-alert, so the procedure MatchAlert is called to match a_2 to both AT and $\widehat{AT}$. Matching a_2 to AT is similar to matching a_1 to AT and there is no correlation of a_2 in the attack template AT. Matching a_2 to $\widehat{AT}$ is a bit different. The procedure finds that node $\widehat{X_3}$ is the only node whose alert unit tables contains alert type `BINARY_FAULT`. Therefore, the procedure FindCorrelation is called with a_2 and $\widehat{X_3}$ as inputs. Since $\widehat{X_1}$ is the parent of $\widehat{X_3}$ and a_1 is a matched alert to $\widehat{X_1}$. The procedure sends both a_2 and a_1 to the alert correlator and finds that they are correlated. Therefore, it returns the correlation score of a_1 and a_2. Since the FindCorrelation procedure finds one correlation of a_2 in the attack template $\widehat{AT}$, a_2 is matched to $\widehat{X_3}$ and a belief propagation is performed on $\widehat{AT}$. After this run, the attack template set $\boldsymbol{ATS}$ contains the original AT and updated $\widehat{AT}$.

Later, EDMAND sends another updated `BINARY_FAULT` meta-alert $\widehat{a_2}$ to CAPTAR as shown in Figure 6.18. The anomaly reasoning algorithm finds that $\widehat{a_2}$ is an update to an existing meta-alert a_2. Also, it finds that $\widehat{AT}$ in $\boldsymbol{ATS}$ contains a_2. Therefore, it replaces a_2 with $\widehat{a_2}$ in $\widehat{AT}$ and starts new belief propagations in $\widehat{AT}$. After this run, the causal reasoning engine finds that $BEL_{max}(\widehat{AT})$ exceeds the predefined threshold. Therefore, the

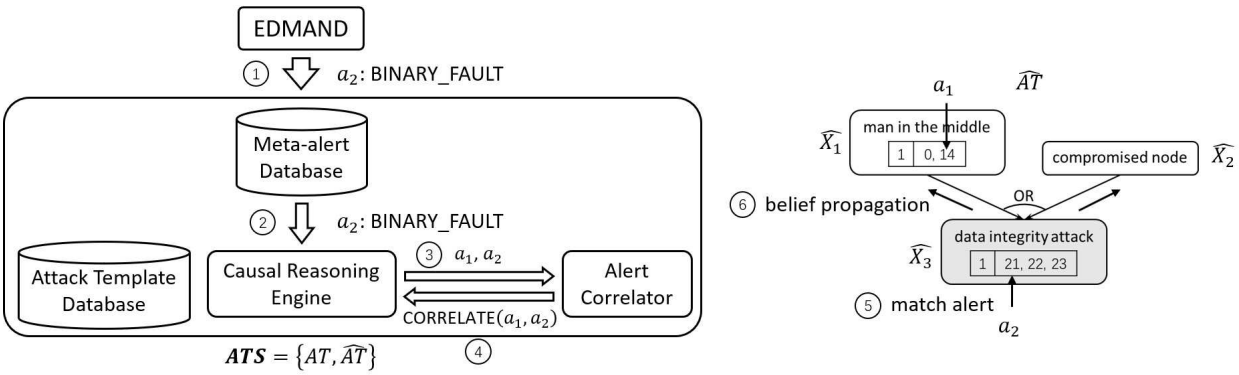


Figure 6.17: Algorithm run upon receiving meta-alert a_2

engine outputs $\widehat{AT}$ to the operator. The operator can see from $\widehat{AT}$ that there is a data integrity attack going on and the attacker first launched a man in the middle attack (MITM) to achieve that. The two matched alerts a_1 and $\widehat{a_2}$ can also be used for more detailed analysis by the operator.

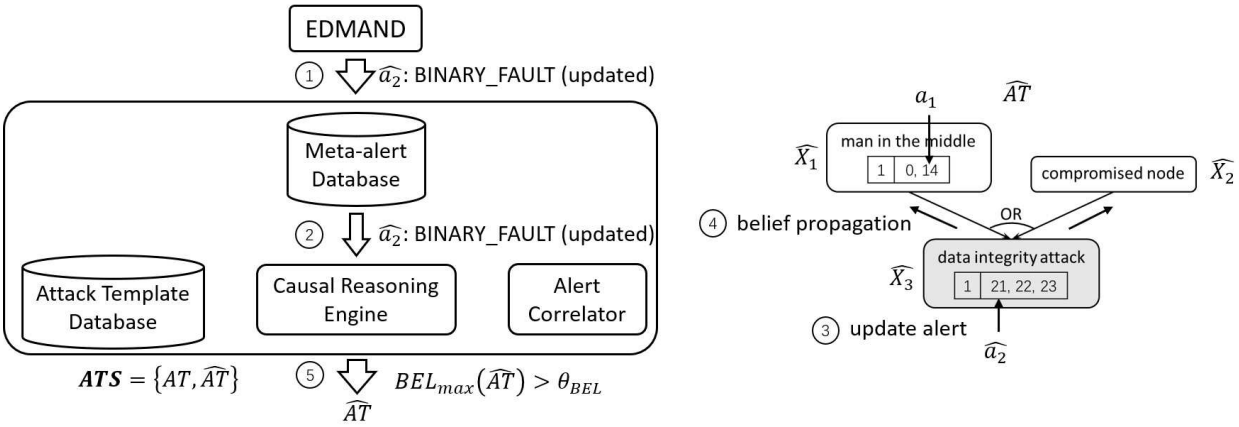


Figure 6.18: Algorithm run upon receiving meta-alert $\widehat{a_2}$

6.5 PERFORMANCE EVALUATION

In this section, we evaluate the anomaly reasoning ability of CAPTAR via three simulated attack scenarios. We implement a prototype of CAPTAR and reuse our prototype of EDMAND described in Chapter 5. The baseline traffic is 14 days of simulated DNP3 traffic of one control center communicating with 10 remote terminal units (RTUs). More details of the baseline traffic can be found in Section 5.6. We create three attack templates representing three common attacks in SCADA networks: TCP SYN flood, data integrity attack, and command injection.

- *TCP SYN flood*: The attack template for TCP SYN flood is shown in Figure 6.19. The attacker starts by an IP address scan to find out the active IP addresses in the subnet. Then the TCP SYN flood is conducted by sending a succession of SYN requests to the target with spoofed source addresses.

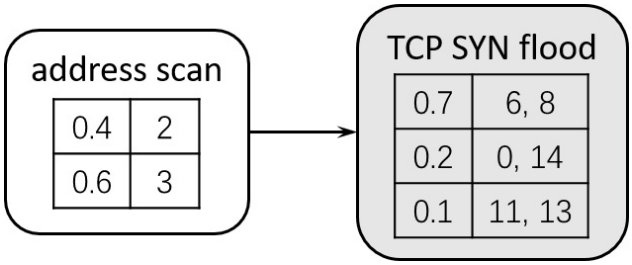


Figure 6.19: TCP SYN flood

- *Data integrity attack*: The attack template for data integrity attack is shown in Figure 6.20. The attacker first either launches a man-in-the-middle attack or compromises some field devices. The measurement data sent back to the control center are then tampered to mislead the control system.

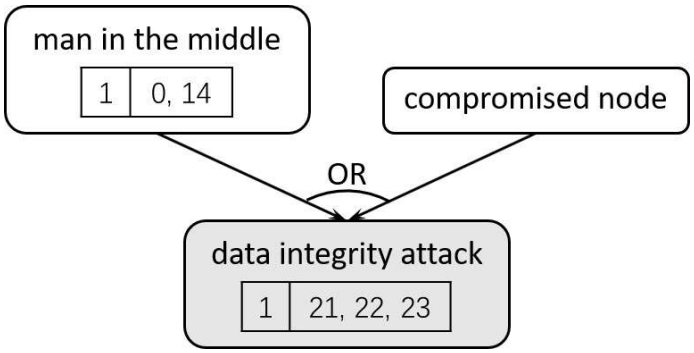


Figure 6.20: Data integrity attack

- *Command injection*: The attack template for command injection is shown in Figure 6.21. The attacker first either launches a man-in-the-middle attack or conducts an IP address scan followed by a service scan. Malicious control commands are then injected into the packets to attack the substations.

In our evaluation, we launch the above three attacks in our simulated SCADA network. CAPTAR together with EDMAND are able to identify and differentiate all three attacks. Moreover, the output of CAPTAR gives the operator a better idea of the likelihood of each attack step even if there is no direct alert representation of the step. For example, the attack

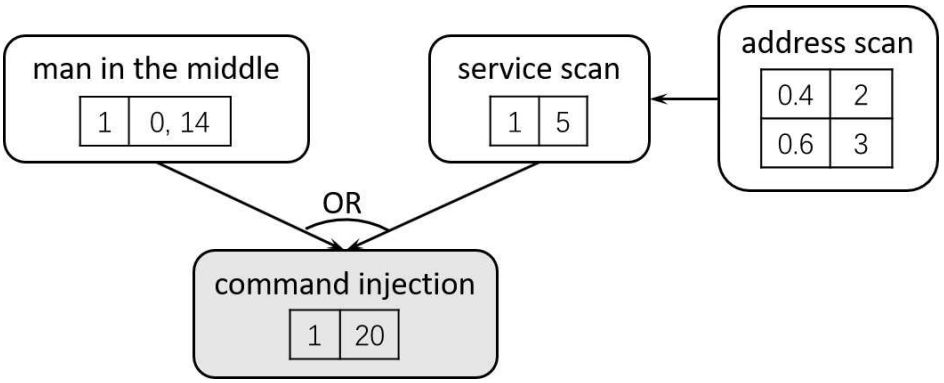


Figure 6.21: Command injection

step of "compromised node" in the data integrity attack has no detectable alert by EDMAND (for now). However, CAPTAR can still infer the high chance of existence of a compromised node if it sees the existence of the "data integrity attack" consequence node and the absence of the "man in the middle" node. Notice that the expressiveness of attack templates can be improved by increasing the number of meta-alert types that can be triggered by EDMAND. CAPTAR can also reason about alerts not from EDMAND as long as they are preprocessed to follow the same format.

We now briefly calculate the time complexity of the anomaly reasoning algorithm. We start by estimating the time complexity of the FindCorrelation procedure. Let us assume M to be the number of meta-alerts in the database. In the worst case, the FindCorrelation needs to correlate the input meta-alert with all other meta-alerts. Since the time complexity of correlating a pair of meta-alerts is constant, the FindCorrelation procedure has a $O(M)$ time complexity. Let us assume the maximum number of nodes in any attack template is N and L is the number of attack templates in the database. In the MatchAlert procedure, the first 'for' loop needs to go over every node in the template in the worst case, which has a time complexity of $O(MN)$. The belief propagation is $O(N)$, and N_{pot} has N nodes in the worst case. So the rest of the procedure has a time complexity of $O(N^2)$. The total time complexity of MatchAlert is therefore $O(MN+N^2)$. In the anomaly reasoning algorithm, the maximum attack template number is KL. It can be easily derived that the time complexity of the algorithm is $O(KLN(M+N))$. Usually, we have $M \gg N$, so the anomaly reasoning algorithm has an estimated time complexity of $O(KLMN)$ in the worst case. K and N are usually less than 10. L should be several dozens. M is also limited to dozens or hundreds due to the alert aggregation and removing of stale meta-alerts from the database. Therefore, the total time complexity of the algorithm is reasonable. And notice that the frequency CAPTAR runs the anomaly reasoning algorithm is decided by the

frequency that EDMAND sends meta-alerts. As mentioned in 5.4.2, EDMAND sends meta-alerts in a periodic manner only if there are updates to those meta-alerts in the latest period. So the sending rate of meta-alerts by EDMAND is also limited. Therefore, CAPTAR is able to satisfy the real-time anomaly reasoning need for those meta-alerts.

To give a better understanding of the time overhead of CAPTAR, we measure the time to run the FindCorrelation procedure, the belief propagation, and the anomaly reasoning algorithm for the three attack scenarios on a Ubuntu 16.04 desktop with 12 Intel Xeon 3.60GHz CPUs and 16GB memory. For each attack scenario, we run CAPTAR on the entire traffic set including the corresponding attack and calculate the average and standard deviation in millisecond of the time overheads for FindCorrelation, belief propagation, and the anomaly reasoning algorithm. We also record the sample number, which is the number of time FindCorrelation, belief propagation, and the anomaly reasoning algorithm have been performed. The results are shown in Table 6.6. We can see that the time overheads are definitely small enough to satisfy the real-time reasoning requirement of the meta-alerts. Note that the average time to run the FindCorrelation procedure and the anomaly reasoning algorithm varies a lot across different attack scenarios. This is because the time overheads of FindCorrelation and the anomaly reasoning algorithm depend on the number of meta-alert M as we described previously. And those three attack scenarios generate 104(TCP SYN flood), 7(data integrity attack), and 26(command injection) meta-alerts respectively. This results in the different time overheads of FindCorrelation and the anomaly reasoning algorithm for them. Another fact is that all the time overheads have relatively high standard deviation. This is mainly due to the change to meta-alert number in the meta-alert database during the attack. As the attack continues, the number of meta-alerts in the database increases, and so do the time overheads for FindCorrelation, belief propagation, and the anomaly reasoning algorithm.

Attack	FindCorrelation			Belief Propagation			Anomaly Reasoning		
	avg	std	num	avg	std	num	avg	std	num
TCP SYN flood	7.60	4.58	64	0.21	0.12	64	41.39	28.90	122
Data integrity attack	0.48	0.37	4	0.10	0.04	4	19.76	48.72	12
Command injection	2.65	1.61	25	0.14	0.02	25	13.95	34.52	40

Table 6.6: Average(avg in ms), standard deviation(std in ms), and sample number(num) of time overhead for FindCorrelation, belief propagation, and the anomaly reasoning algorithm

6.6 CONCLUSION

In this chapter, we propose a causal-polytree-based anomaly reasoning framework for SCADA networks, named CAPTAR. CAPTAR takes the meta-alerts from EDMAND and performs alert correlation and attack plan recognition. Experiments using a prototype of CAPTAR and simulated traffic show that CAPTAR is able to detect and differentiate various attack scenarios in a real-time manner. The generated reasoning results can provide the operators with a high-level view of the security state of the protected SCADA network.

CHAPTER 7: CONCLUSIONS AND DISCUSSION

In this chapter, I first conclude this thesis by summarizing each work I have done. After that, I will discuss how domain specific is this thesis and how it can be generalized to a broader domain.

7.1 CONCLUSIONS

In this thesis, I use an integrated approach of edge-cloud design, real-time data operations, and causal security analysis and propose four frameworks to help to guarantee the situational awareness of Smart Grid. One of them protects situational awareness in WAMS by using online data compression to reduce the huge and increasing volume of data in the communication networks to avoid congestions. The other three of them use anomaly detection and causal anomaly reasoning to enhance security of SCADA systems and thus guarantee situational awareness. Note that although I focus only on WAMS and SCADA systems in Smart Grid in this thesis, the proposed frameworks could potentially be utilized to help with situational awareness of similar industrial control systems in other large-scale distributed critical infrastructure systems (e.g., oil/gas pipelines and refineries, water distribution and treatment) after minor adaptations.

7.1.1 OLAF: Operation-Level Traffic Analyzer Framework for SCADA System

The current SCADA systems in Smart Grid are facing increasing security risks. To provide end-to-end security against both external and internal attacks, both the end host devices and the network need to be secured. In the current architecture, the control centers are responsible for both the device status analysis and network traffic analysis. Since the control center needs to wait for the data from measurement devices before doing any processing and analysis, it is increasingly hard to provide up-to-date situational awareness as the number of deployed measurement devices grows. To address this issue, in Chapter 3, I propose OLAF, an edge-based, extensible, and efficient operation-level traffic analyzer. OLAF resides in substations of SCADA systems and is able to provide preliminary but more prompt situational awareness by performing network traffic analysis and device status analysis at the edge. Experimental results are encouraging by showing strong anomaly detection ability and low time overhead of OLAF.

7.1.2 ISAAC: Intelligent Synchrophasor Data Real-Time Compression Framework for WAMS

The huge and rapidly increasing data volume in WAMS imposes a heavy burden on the communication and storage systems and could result in frequent and severe congestion if not handled carefully. The situational awareness of the system could suffer a lot from the extremely long delays or high packet loss rates that follow the congestion. In Chapter 4, I propose ISAAC, an intelligent synchrophasor data real-time compression framework for WAMS to be deployed at the edge of WAMS. Based on a combination of PCA and DCT techniques, ISAAC is able to mitigate the burden on communication systems laid by the huge synchrophasor data volume while satisfying the requirements of real-time WAMS applications. A disturbance detector is utilized to identify disturbance data and satisfy its stricter delay and accuracy requirements. ISAAC can achieve good compression ratios while maintaining satisfying delay and accuracy for the reconstructed data. The performance of ISAAC is validated by experiments based on real synchrophasor data.

7.1.3 EDMAND: Edge-Based Multi-Level Anomaly Detection for SCADA Networks

The light-weighted operation-level traffic analyzer, named OLAF, in Chapter 3 provides preliminary analysis of SCADA. That is not enough to guarantee situational awareness and a more thorough monitoring and analysis is required. Based on different analysis granularity, data in SCADA network traffic generally can be divided into three levels: transport level, operation level, and content level. Monitoring and event detection of only one or two of the three levels is not enough to detect and reason about attacks in all three levels. In Chapter 5, I develop EDMAND, an edge-based multi-level anomaly detection framework for SCADA networks. EDMAND resides in remote substations of SCADA systems and monitors network traffic at flow level, operation level, and content level. Distinct data characteristics are taken into consideration when selecting anomaly detection method for each level. When anomalies are detected, EDMAND generates, aggregates, and prioritizes alerts and sends them to control centers. Moreover, the concept of confidence score is introduced into the anomaly detection process and confidence scores are assigned to generated alerts. The performance of EDMAND is validated by synthetic DNP3 traffic with various anomalies injected.

7.1.4 CAPTAR: Causal-Polytree-based Anomaly Reasoning for SCADA Networks

My work EDMAND, described in Chapter 5, resides at the edges of the SCADA network, detects anomalies at multiple levels of the network, and sends aggregated and prioritized

meta-alerts to the control center. However, only knowing what anomalies are happening in the system without understanding why they happen is definitely not enough to guarantee situational awareness. There is a need for an efficient system to correlate the alerts in an intelligent manner and match temp to potential attack(s) based on domain knowledge to discover attack strategies. In Chapter 6, I present a causal-polytree-based anomaly reasoning framework for SCADA networks, named CAPTAR. CAPTAR takes the meta-alerts from EDMAND and performs alert correlation and attack plan recognition. Experiments, using a prototype of CAPTAR and simulated traffic, show that CAPTAR is able to detect and differentiate across various attack scenarios in real time. The generated reasoning results can provide the operators with a high-level view of the security state of the protected SCADA network.

7.2 DISCUSSION

The four frameworks in this thesis (OLAF, ISAAC, EDMAND, CAPTAR) are designed to provide situational awareness to Smart Grid and are domain specific. Quite an amount of domain-specific knowledge is utilized during the development of the frameworks. For example, one important prerequisite that allows EDMAND to use statistics such as mean and standard deviation to perform anomaly detection is that the traffic in SCADA networks is mostly periodic. The same approach cannot work for the Internet which has much more irregular and bursty traffic. Also, the characteristics of measurement data utilized by EDMAND to do content-level anomaly detection is only valid for this specific domain. CAPTAR is designed specifically for Smart Grid as well with the parameters in the alert correlator set by domain experts and attack template database containing typical attacks for this domain.

Although the four frameworks in this thesis are designed specifically for Smart Grid, the contribution of this thesis is more than proposing approaches to provide situational awareness to Smart Grid. Some methodologies used by this thesis can be applied to broader contexts. For example, ISAAC identifies the different compression requirements for normal traffic and traffic with disturbances, and applies different compression methods. The the compression technique of reusing the transformation matrix in Principal component analysis can also be generalized to other domains. EDMAND benefits from the idea of dividing data to multiple levels by different analysis granularity and applying appropriate mechanism to each level. Introducing the concept of confidence score to anomaly detection and assigning confidence scores to alerts is another valuable design decision that can be used by other domains. The way to differentiate data measurements into different classes using Bayesian inference is not

limited to the specific scenarios. Actually, both the meta-alert priority score calculation in EDMAND and the alert correlation in CAPTAR reuse the same concept. By using causal reasoning, CAPTAR is able to combine weak indications (indirect evidence) of anomalies, represented by meta-alerts, and derive stronger indications. The causal polytrees utilizing "noisy-OR" and "noisy-AND" models, the auxiliary child to represent indirect evidence, and the belief propagation on those polytress can all be adapted to different use cases. And the anomaly reasoning algorithm is also not domain-specific. All these methodologies are not limited to the specific domain of Smart Grid and can be applied to broader domains.

For example, the aforementioned methodologies can be generalized to the domain of Internet of Things (IoT). Providing situational awareness in IoT is also of great importance. Security and data volume are issues in IoT as well. To adapt frameworks in this thesis to the domain of IoT, several changes need to be made on the frameworks and some major ones are listed as follows.

- The parsers and data extractors in this thesis only support several communication protocols in Smart Grid (e.g. Modbus, DNP3). They need to be modified to support communication protocols in IoT.

- The attack templates in CAPTAR correspond to attacks commonly seen in SCADA networks. However, the common attacks in SCADA might not be the same with the common attacks in IoT. New attack templates for the IoT domain need to be created by experts.

- In EDMAND, the different characteristics of data traffic in different levels are utilized to select appropriate anomaly detection mechanisms. The characteristics of traffic in IoT are different from those in SCADA. The new characteristics need to be reconsidered to select appropriate anomaly detection mechanisms for IoT domain.

- The ontology of alerts in this thesis is domain-specific. The alerts have fields (even protocol specific fields) typical to SCADA and the alert types are also defined by domain knowledge. The ontology needs to be adapted to IoT domain and alert types need to be redefined to reflect common anomalies in IoT.

- Various domain specific parameters in the frameworks need to be reset by experts from IoT domain. For example, in the alert correlator of CAPTAR, the prior probabilities of different alert types and the elements in the conditional probability table are all domain-specific and have new values in the IoT domain.

- The adapted frameworks also need to meet the time overhead requirements in IoT which are different from those in Smart Grid. For example, the current time overheads of EDMAND and CAPTAR are good enough for SCADA networks. If IoT applications have stricter time requirements, the anomaly detection and anomaly reasoning algorithms might need to be simplified or performed in a more efficient manner to guarantee lower time overheads.

CHAPTER 8: LESSONS LEARNED AND FUTURE DIRECTIONS

In this chapter, I first share some of the lessons I learned while working on this thesis. Then some preliminary thoughts of potential future research directions are given.

8.1 LESSONS LEARNED

My Ph.D. study was a tough but fruitful journey. I have learned a lot from all the research I have done as well as working on this thesis and I want to share several lessons I have learned.

The first lesson is that getting access to real world traffic for cyber-physical systems such as the Smart Grid is challenging. Since cyber-physical systems are usually critical infrastructures, utilities are not willing to share their traffic traces due to security and privacy concerns. Acquiring normal traffic is very hard and getting access to real traffic with attacks is almost impossible. Therefore, sometimes researchers have no alternative but to use and work with simulated data. Developing better simulators and techniques that help utilities to share their data in secure manner could be topics worth exploring.

The second lesson is that learning and understanding domain knowledge, and collaborating with domain experts is beneficial and necessary for delivering impactful research results. Developing approaches to provide situational awareness for Smart Grid requires a good understanding of cyber systems (e.g., SCADA, WAMS), devices (e.g., PMUs, MTUs, RTUs), communication protocols (e.g., Modbus, DNP3), measurement characteristics, domain specific attacks, etc. Knowing the physical model and physical laws in Smart Grid can also help with identifying the normal and abnormal behaviors in the system. Working closely with domain experts ensures that research has practical value and can be deployed in real systems.

The third lesson is that extending the breadth of knowledge is of great importance to do interdisciplinary research. For example, in my thesis, there is an integration of knowledge from various areas including system design, data analytic, security, compression techniques, real-time systems, edge computing, Bayesian inference, causal analysis, etc. Applying well-known techniques from other domains and adapting them to my own research area was a good way to come up with novel research ideas.

Last but not the least, accessing and identifying the limits of my work was very helpful in finding new research directions. For example, after I developed OLAF, I found that OLAF was not able to detect any anomaly in measurement data. Also, sometimes the frequency of alerts was too high, which meant difficulty for operators to learn useful information from

the alerts. Those limits of OLAF actually inspired the development of both EDMAND and CAPTAR. Another example is that although EDMAND's time overhead is good enough for SCADA networks, it is still too large for systems with much higher data sampling frequencies such as WAMS. Therefore, developing anomaly detection mechanisms and data structures with higher time efficiency might be an interesting future research direction.

8.2 FUTURE RESEARCH DIRECTIONS

The complexity of Smart Grid keeps growing as new technologies are introduced to it. Therefore, operating Smart Grid will become increasingly challenging and guaranteeing situational awareness will be even more important. In this thesis, I am only able to cover a limited part of the problem and there are many other aspects that remain to be explored. Some potential future research directions that are highly related to this thesis are as follows.

First, I mainly focus on the anomaly detection and anomaly reasoning on SCADA network traffic in OLAF, EDMAND, and CAPTAR. The physical model of the system can also provide potentially useful information for the detection and reasoning of anomalies. Taking the physical model and constraints into consideration could help with the analysis of measurement data and system state. Therefore, one direction of future research is to utilize a hybrid approach of cyber and physical models to provide comprehensive security analysis of Smart Grid.

Second, after the attack plan is recognized as discussed in CAPTAR, the operator of SCADA networks still needs to decide how to react to it. It is beneficial if some potential consequences and responses could be provided together with the inferred attack. Therefore, how to give useful suggestions for countermeasures to attacks, given the inferred attack plan and current state of the system, could be an interesting research problem.

Third, various mechanisms to help to create attack templates could be explored. Specification language could be designed to help domain experts to transfer their domain knowledge into attack templates. Also, the structural causal models and various tools mentioned in [103] could be utilized to find the causal relationship of attack steps from existing attack traces.

Fourth, due to the reasons I mentioned in Section 8.1, real traffic is hard to acquire. Some frameworks in this thesis are evaluated by simulated traffic. If in the future access to real traffic or even real attack traces could be obtained, it is worth evaluating those frameworks again by real traffic data. If further collaboration with utilities can be achieved, deploying those frameworks to real systems and evaluating the real-time performance of them could be pretty interesting.

Fifth, as I discussed in Section 7.2, frameworks in this thesis have the potential to be generalized to broader areas. Therefore, adapting those frameworks and reapplying them to other domains such as general IoT networks and data center networks could inspire new research projects.

Last but not least, SCADA and WAMS are used together in many Smart Grids today. The interdependency of these two complex systems could create entirely new security and managing problems. For example, the high data rate in WAMS introduces stricter time requirements for security protection mechanisms and more time-efficient approaches need to be developed. Targeting those new problems by considering these two systems as a whole is worth researchers' attention.

REFERENCES

[1] S. M. Amin, "Smart grid: Overview, issues and opportunities. advances and challenges in sensing, modeling, simulation, optimization and control," *European Journal of Control*, vol. 17, no. 5-6, pp. 547–567, 2011.

[2] M. Amin, "Security challenges for the electricity infrastructure," *Computer*, vol. 35, no. 4, pp. supl8–supl10, 2002.

[3] M. Amin, "North America's electricity infrastructure: Are we ready for more perfect storms?" *IEEE Security & Privacy*, vol. 99, no. 5, pp. 19–25, 2003.

[4] M. Amin and J. Stringer, "The electric power grid: Today and tomorrow," *MRS bulletin*, vol. 33, no. 4, pp. 399–407, 2008.

[5] S. M. Amin, "US grid gets less reliable [The Data]," *IEEE Spectrum*, vol. 48, no. 1, pp. 80–80, 2011.

[6] T. Vijayapriya and D. P. Kothari, "Smart grid: an overview," *Smart Grid and Renewable Energy*, vol. 2, no. 04, p. 305, 2011.

[7] A. A. Cardenas, S. Amin, and S. Sastry, "Secure control: Towards survivable cyber-physical systems," in *Distributed Computing Systems Workshops, 2008. ICDCS'08. 28th International Conference on*. IEEE, 2008, pp. 495–500.

[8] C. Greer, D. A. Wollman, D. E. Prochaska, P. A. Boynton, J. A. Mazer, C. T. Nguyen, G. J. FitzPatrick, T. L. Nelson, G. H. Koepke, A. R. Hefner Jr et al., "Nist framework and roadmap for smart grid interoperability standards, release 3.0," Tech. Rep., 2014.

[9] A. Mavridou and M. Papa, "A situational awareness architecture for the smart grid," in *Global Security, Safety and Sustainability & e-Democracy*. Springer, 2011, pp. 229–236.

[10] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[11] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "On the integration of cloud computing and internet of things," in *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*. IEEE, 2014, pp. 23–30.

[12] N. R. Council et al., *Modeling human and organizational behavior: Application to military simulations*. National Academies Press, 1998.

[13] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[14] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*. ACM, 2015, pp. 37–42.

[15] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, 2017.

[16] F. Cicirelli, G. Fortino, A. Guerrieri, G. Spezzano, and A. Vinci, "Edge enabled development of Smart Cyber-Physical Environments," in *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 003 463–003 468.

[17] K. Stouffer, J. Falco, and K. Scarfone, "Guide to industrial control systems (ICS) security," *NIST special publication*, vol. 800, no. 82, pp. 16–16, 2011.

[18] W. Ren, S. Granda, T. Yardley, K.-S. Lui, and K. Nahrstedt, "OLAF: Operation-level traffic analyzer framework for Smart Grid," in *Smart Grid Communications (SmartGridComm), 2016 IEEE International Conference on*. IEEE, 2016, pp. 551–556.

[19] W. Ren, T. Yardley, and K. Nahrstedt, "ISAAC: Intelligent Synchrophasor Data Real-Time Compression Framework for WAMS," in *Smart Grid Communications (SmartGridComm), 2017 IEEE International Conference on*. IEEE, 2017.

[20] W. Ren, T. Yardley, and K. Nahrstedt, "EDMAND: Edge-Based Multi-Level Anomaly Detection for SCADA Networks," in *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE, 2018, pp. 1–7.

[21] B. Liscouski and W. Elliot, "Final report on the august 14, 2003 blackout in the united states and canada: Causes and recommendations," *A report to US Department of Energy*, vol. 40, no. 4, p. 86, 2004.

[22] G. Clarke, D. Reynders, and E. Wright, *Practical modern SCADA protocols: DNP3, 60870.5 and related systems*. Newnes, 2004.

[23] I. Modicon, "Modicon modbus protocol reference guide," *North Andover, Massachusetts*, pp. 28–29, 1996.

[24] Specification, Modbus Application Protocol, "V1. 1b3," *Hopkinton: Modbus Organization, Inc., April*, vol. 26, 2012.

[25] "IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3)," *IEEE Std 1815-2012 (Revision of IEEE Std 1815-2010)*, pp. 1–821, Oct 2012.

[26] K. Curtis, "A DNP3 protocol primer," *DNP User Group*, vol. 2005, 2005.

[27] T. McGregor, H.-W. Braun, and J. Brown, "The NLANR Network Analysis Infrastructure," *Communications Magazine, IEEE*, vol. 38, no. 5, pp. 122–128, May 2000.

[28] W. Erhard, M. Gutzmann, and H. Libati, "Network traffic analysis and security monitoring with UniMon," in *High Performance Switching and Routing, 2000. ATM 2000. Proceedings of the IEEE Conference on*, 2000, pp. 439–446.

[29] D. Keim, F. Mansmann, J. Schneidewind, and T. Schreck, "Monitoring Network Traffic with Radial Traffic Analyzer," in *Visual Analytics Science And Technology, 2006 IEEE Symposium On*, Oct 2006, pp. 123–128.

[30] M. Rahman, Z. Khalib, and R. Ahmad, "A portable network traffic analyzer," in *Electronic Design, 2008. ICED 2008. International Conference on*, Dec 2008, pp. 1–6.

[31] I. N. Fovino, A. Coletta, A. Carcano, and M. Masera, "Critical state-based filtering system for securing SCADA network protocols," *IEEE Transactions on industrial electronics*, vol. 59, no. 10, pp. 3943–3950, 2012.

[32] B.-K. Kim, D.-H. Kang, J.-C. Na, and T.-M. Chung, "Abnormal traffic filtering mechanism for protecting ICS networks," in *2016 18th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 2016, pp. 436–440.

[33] S. Parthasarathy and D. Kundur, "Bloom filter based intrusion detection for smart grid SCADA," in *Electrical & Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on*. IEEE, 2012, pp. 1–6.

[34] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes, "Using model-based intrusion detection for SCADA networks," in *Proceedings of the SCADA security scientific symposium*, vol. 46. Citeseer, 2007, pp. 1–12.

[35] J. M. Beaver, R. C. Borges-Hink, and M. A. Buckner, "An evaluation of machine learning methods to detect malicious SCADA communications," in *Machine Learning and Applications (ICMLA), 2013 12th International Conference on*, vol. 2. IEEE, 2013, pp. 54–59.

[36] L. A. Maglaras and J. Jiang, "Intrusion detection in SCADA systems using machine learning techniques," in *Science and Information Conference (SAI), 2014*. IEEE, 2014, pp. 626–631.

[37] B. Panja, J. Oros, J. Britton, P. Meharia, and S. Pati, "Intelligent gateway for SCADA system security: A multi-layer attack prevention approach," in *2015 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*. IEEE, 2015, pp. 1–6.

[38] D. Hadziosmanovic, D. Bolzoni, S. Etalle, and P. Hartel, "Challenges and opportunities in securing industrial control systems," in *Complexity in Engineering (COMPENG), 2012*. IEEE, 2012, pp. 1–6.

[39] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23, pp. 2435–2463, 1999.

[40] M. Patel, S. Aivaliotis, E. Ellen et al., "Real-time application of synchrophasors for improving reliability," *NERC Report, Oct*, 2010.

[41] F. Zhang, L. Cheng, X. Li, Y. Sun, W. Gao, and W. Zhao, "Application of a real-time data compression and adapted protocol technique for WAMS," *IEEE Transactions on Power Systems*, vol. 30, no. 2, pp. 653–662, 2015.

[42] "Synchrophasor technology fact sheet," October 2014. [Online]. Available: https://www.naspi.org/File.aspx?fileID=1326

[43] R. Klump, P. Agarwal, J. E. Tate, and H. Khurana, "Lossless compression of synchronized phasor measurements," in *Power and Energy Society General Meeting, 2010 IEEE*. IEEE, 2010, pp. 1–7.

[44] M. Ringwelski, C. Renner, A. Reinhardt, A. Weigel, and V. Turau, "The hitchhiker's guide to choosing the compression algorithm for your smart meter data," in *Energy Conference and Exhibition (ENERGYCON), 2012 IEEE International*. IEEE, 2012, pp. 935–940.

[45] J. Kraus, P. Štěpán, and L. Kukačka, "Optimal data compression techniques for smart grid and power quality trend data," in *Harmonics and Quality of Power (ICHQP), 2012 IEEE 15th International Conference on*. IEEE, 2012, pp. 707–712.

[46] J. Ning, J. Wang, W. Gao, and C. Liu, "A wavelet-based data compression technique for smart grid," *IEEE Transactions on Smart Grid*, vol. 2, no. 1, pp. 212–218, 2011.

[47] M. Wang, J. H. Chow, P. Gao, X. T. Jiang, Y. Xia, S. G. Ghiocel, B. Fardanesh, G. Stefopolous, Y. Kokai, N. Saito et al., "A low-rank matrix approach for the analysis of large amounts of power system synchrophasor data," in *System Sciences (HICSS), 2015 48th Hawaii International Conference on*. IEEE, 2015, pp. 2637–2644.

[48] P. H. Gadde, M. Biswal, S. Brahma, and H. Cao, "Efficient Compression of PMU Data in WAMS," *IEEE Transactions on Smart Grid*, vol. 7, no. 5, pp. 2406–2413, Sept 2016.

[49] J. C. S. de Souza, T. M. L. Assis, and B. C. Pal, "Data Compression in Smart Distribution Systems via Singular Value Decomposition," *IEEE Transactions on Smart Grid*, vol. 8, no. 1, pp. 275–284, 2017.

[50] M. Chenine, K. Zhu, and L. Nordstrom, "Survey on priorities and communication requirements for PMU-based applications in the Nordic Region," in *PowerTech, 2009 IEEE Bucharest*. IEEE, 2009, pp. 1–8.

[51] D. E. Bakken, A. Bose, C. H. Hauser, D. E. Whitehead, and G. C. Zweigle, "Smart generation and transmission with coherent, real-time data," *Proceedings of the IEEE*, vol. 99, no. 6, pp. 928–951, 2011.

[52] K. Khandeparkar, K. Ramamritham, R. Gupta, A. Kulkarni, G. Gajjar, and S. Soman, "Timely Query Processing in Smart Electric Grids: Algorithms and Performance," in *Proceedings of the 2015 ACM Sixth International Conference on Future Energy Systems*. ACM, 2015, pp. 161–170.

[53] C. Martinez, M. Parashar, J. Dyer, and J. Coroas, "Phasor data requirements for real time wide-area monitoring, control and protection applications," *EIPP White Paper*, vol. 26, p. 8, 2005.

[54] "IEEE Guide for Phasor Data Concentrator Requirements for Power System Protection, Control, and Monitoring," *IEEE Std C37.244-2013*, pp. 1–65, May 2013.

[55] A. Moga and T. Locher, "Scalable and reliable monitoring for power systems," in *2015 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, Nov 2015, pp. 259–264.

[56] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002.

[57] scikit-learn developers, "sklearn.decomposition module," http://scikit-learn.org/stable/modules/classes.html#module-sklearn.decomposition.

[58] K. R. Rao and P. Yip, *Discrete cosine transform: algorithms, advantages, applications*. Academic press, 2014.

[59] scipy community, "scipy.fftpack.dct," https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.fftpack.dct.html.

[60] North American Electric Reliability Corporation (NERC), "Prc-002-2 disturbance monitoring and reporting requirements," April 2017.

[61] M. Biswal, S. M. Brahma, and H. Cao, "Supervisory Protection and Automated Event Diagnosis Using PMU Data," *IEEE Transactions on Power Delivery*, vol. 31, no. 4, pp. 1855–1863, Aug 2016.

[62] R. W. G. C. for Electricity Innovation, "Microgrid Project at IIT," http://iitmicrogrid.net/microgrid.aspxl.

[63] K. Zhu, M. Chenine, L. Nordström, S. Holmström, and G. Ericsson, "An empirical study of synchrophasor communication delay in a utility TCP/IP network," *International Journal of Emerging Electric Power Systems*, vol. 14, no. 4, pp. 341–350, 2013.

[64] M. Chenine, K. Zhu, and L. Nordstrom, "Survey on priorities and communication requirements for PMU-based applications in the Nordic Region," in *PowerTech, 2009 IEEE Bucharest*. IEEE, 2009, pp. 1–8.

[65] D. E. Bakken, A. Bose, C. H. Hauser, D. E. Whitehead, and G. C. Zweigle, "Smart generation and transmission with coherent, real-time data," *Proceedings of the IEEE*, vol. 99, no. 6, pp. 928–951, 2011.

[66] N. Falliere, L. O. Murchu, and E. Chien, "W32. stuxnet dossier," *White paper, Symantec Corp., Security Response*, vol. 5, no. 6, p. 29, 2011.

[67] D. U. Case, "Analysis of the cyber attack on the Ukrainian power grid," *Electricity Information Sharing and Analysis Center (E-ISAC)*, 2016.

[68] L. A. Maglaras and J. Jiang, "Intrusion detection in scada systems using machine learning techniques," in *Science and Information Conference (SAI), 2014*. IEEE, 2014, pp. 626–631.

[69] R. Udd, M. Asplund, S. Nadjm-Tehrani, M. Kazemtabrizi, and M. Ekstedt, "Exploiting bro for intrusion detection in a SCADA system," in *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*. ACM, 2016, pp. 44–51.

[70] I. N. Fovino, A. Coletta, A. Carcano, and M. Masera, "Critical state-based filtering system for securing SCADA network protocols," *IEEE Transactions on industrial electronics*, vol. 59, no. 10, pp. 3943–3950, 2012.

[71] Y. Yang, K. McLaughlin, T. Littler, S. Sezer, B. Pranggono, and H. Wang, "Intrusion detection system for IEC 60870-5-104 based SCADA networks," in *Power and Energy Society General Meeting (PES), 2013 IEEE*. IEEE, 2013, pp. 1–5.

[72] J. M. Beaver, R. C. Borges-Hink, and M. A. Buckner, "An evaluation of machine learning methods to detect malicious SCADA communications," in *Machine Learning and Applications (ICMLA), 2013 12th International Conference on*, vol. 2. IEEE, 2013, pp. 54–59.

[73] H. Lin, A. Slagell, C. Di Martino, Z. Kalbarczyk, and R. K. Iyer, "Adapting bro into scada: building a specification-based intrusion detection system for the dnp3 protocol," in *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*. ACM, 2013, p. 5.

[74] H. R. Ghaeini and N. O. Tippenhauer, "Hamids: Hierarchical monitoring intrusion detection system for industrial control systems," in *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*. ACM, 2016, pp. 103–111.

[75] W. Gao, T. Morris, B. Reaves, and D. Richey, "On SCADA control system command and response injection and intrusion detection," in *eCrime Researchers Summit (eCrime), 2010*. IEEE, 2010, pp. 1–9.

[76] J. Nivethan and M. Papa, "A SCADA Intrusion Detection Framework that Incorporates Process Semantics," in *Proceedings of the 11th Annual Cyber and Information Security Research Conference*. ACM, 2016, p. 6.

[77] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes, "Using model-based intrusion detection for SCADA networks," in *Proceedings of the SCADA security scientific symposium*, vol. 46. Citeseer, 2007, pp. 1–12.

[78] N. Saunders, B. Khanna, and T. Collins, "Real-time situational awareness for critical infrastructure protection," in *Smart Grid Communications (SmartGridComm), 2015 IEEE International Conference on*. IEEE, 2015, pp. 151–156.

[79] J. Verba and M. Milvich, "Idaho national laboratory supervisory control and data acquisition intrusion detection system (SCADA IDS)," in *Technologies for Homeland Security, 2008 IEEE Conference on*. IEEE, 2008, pp. 469–473.

[80] Y. Yang, K. McLaughlin, S. Sezer, T. Littler, E. G. Im, B. Pranggono, and H. Wang, "Multiattribute SCADA-specific intrusion detection system for power networks," *IEEE Transactions on Power Delivery*, vol. 29, no. 3, pp. 1092–1102, 2014.

[81] N. Goldenberg and A. Wool, "Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems," *International Journal of Critical Infrastructure Protection*, vol. 6, no. 2, pp. 63–75, 2013.

[82] N. Erez and A. Wool, "Control variable classification, modeling and anomaly detection in Modbus/TCP SCADA systems," *International Journal of Critical Infrastructure Protection*, vol. 10, pp. 59–70, 2015.

[83] S. Ponomarev and T. Atkison, "Industrial control system network intrusion detection by telemetry analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 2, pp. 252–260, 2016.

[84] R. R. R. Barbosa, "Anomaly detection in SCADA systems: a network based approach," 2014.

[85] F. Cao, M. Estert, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *Proceedings of the 2006 SIAM international conference on data mining*. SIAM, 2006, pp. 328–339.

[86] "Cyber-Physical Experimentation Environment for RADICS." [Online]. Available: https://iti.illinois.edu/research/energy-systems/cyber-physical-experimentation-environment-radics-ceer

[87] X. Qin, "A probabilistic-based framework for infosec alert correlation," Ph.D. dissertation, Georgia Institute of Technology, 2005.

[88] "IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3)," *IEEE Std 1815-2012 (Revision of IEEE Std 1815-2010)*, pp. 1–821, Oct 2012.

[89] M. Hadley and K. Huston, "Secure scada communication protocol performance test results," *Pacific Northwest National Laboratory (August 2007)*, 2007.

[90] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2001, pp. 54–68.

[91] F. Cuppens, "Managing alerts in a multi-intrusion detection environment," in *acsac*. IEEE, 2001, p. 0022.

[92] S. Staniford, J. A. Hoagland, and J. M. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security*, vol. 10, no. 1-2, pp. 105–136, 2002.

[93] A. Siraj and R. B. Vaughn, "Multi-level alert clustering for intrusion detection sensor data," in *NAFIPS 2005-2005 Annual Meeting of the North American Fuzzy Information Processing Society*. IEEE, 2005, pp. 748–753.

[94] S. Zhang, J. Li, X. Chen, and L. Fan, "Building network attack graph for alert causal correlation," *Computers & security*, vol. 27, no. 5-6, pp. 188–196, 2008.

[95] L. Briesemeister, S. Cheung, U. Lindqvist, and A. Valdes, "Detection, correlation, and visualization of attacks against critical infrastructure systems," in *2010 Eighth International Conference on Privacy, Security and Trust*. IEEE, 2010, pp. 15–22.

[96] Y. Zhai, P. Ning, P. Iyer, and D. S. Reeves, "Reasoning about complementary intrusion evidence," in *20th Annual Computer Security Applications Conference*. IEEE, 2004, pp. 39–48.

[97] Z. Zali, M. R. Hashemi, and H. Saidi, "Real-time attack scenario detection via intrusion detection alert correlation," in *2012 9th International ISC Conference on Information Security and Cryptology*. IEEE, 2012, pp. 95–102.

[98] A. Valdes and K. Skinner, "Adaptive, model-based monitoring for cyber attack detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2000, pp. 80–93.

[99] F. Xuewei, W. Dongxia, H. Minhuan, and S. Xiaoxia, "An approach of discovering causal knowledge for alert correlating based on data mining," in *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*. IEEE, 2014, pp. 57–62.

[100] F. Kavousi and B. Akbari, "A Bayesian network-based approach for learning attack strategies from intrusion alerts," *Security and Communication Networks*, vol. 7, no. 5, pp. 833–853, 2014.

[101] A. A. Ramaki, M. Amini, and R. E. Atani, "RTECA: Real time episode correlation algorithm for multi-step attack scenarios detection," *computers & security*, vol. 49, pp. 206–219, 2015.

[102] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[103] J. Pearl, "The Seven Tools of Causal Inference with Reflections on Machine Learning," Technical Report, Communications of Association for Computing Machinery, Tech. Rep., 2018.